L Number	Hits	Search Text	DB	Time stamp
1	2204	candidate with pair	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:38
2	77105	equivalent with descri$7	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:39
3	69	(candidate with pair) and (equivalent with descri$7)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:42
4	4	((candidate with pair) and (equivalent with descri$7)) and queries and score and threshold	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:54
5	38	(candidate with pair) and (queries with descri$6)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:57
6	12	((candidate with pair) and (queries with descri$6)) and score and threshold	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:49
7	5	@pd<20020201 and (((candidate with pair) and (queries with descri$6)) and score and threshold)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:54
8	8	@pd<20020201 and ((candidate with pair) and (queries with descri$6))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:54
9	82	(candidate with pair) and queries and score and threshold	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:54
10	24	@pd<20020201 and ((candidate with pair) and queries and score and threshold)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:10
11	1	(@pd<20020201 and ((candidate with pair) and queries and score and threshold)) and score and collection and synonym	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:58
12	248	(candidate with pair) and (queries and descri$6)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:57
13	26	(candidate with pair) and (queries and (descri$6 with candidate))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:57
14	3	((candidate with pair) and (queries and (descri$6 with candidate))) and score and collection and synonym	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:07

15	3045	queries with terms	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:08
16	727	(queries with terms) and candidate	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:08
17	103	((queries with terms) and candidate) and collection and threshold and users and calculat$ and (frequen$ with occur$)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:09
18	23	@pd<20020201 and (((queries with terms) and candidate) and collection and threshold and users and calculat$ and (frequen$ with occur$))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:10
20	1	((@pd<20020201 and (((queries with terms) and candidate) and collection and threshold and users and calculat$ and (frequen$ with occur$))) and synonym) and (misspell$ or (spell with check$))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:10
19	8	(@pd<20020201 and (((queries with terms) and candidate) and collection and threshold and users and calculat$ and (frequen$ with occur$))) and synonym	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:16
21	1	((((@pd<20020201 and (((queries with terms) and candidate) and collection and threshold and users and calculat$ and (frequen$ with occur$))) and synonym) and (misspell$ or (spell with check$))) and pair	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:13
22	0	(707/3).ccls. and (@pd<20020201 and ((candidate with pair) and (queries with descri$6)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:17
23	1	(707/5).ccls. and (@pd<20020201 and ((candidate with pair) and (queries with descri$6)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:17
24	0	(707/4).ccls. and (@pd<20020201 and ((candidate with pair) and (queries with descri$6)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 09:17
-	2	5897622.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/25 08:38
-	2	5819265.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 13:26
-	0	584103.apn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 13:58
-	125	document with proper with name	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 13:58

-	34	(document with proper with name) and categor$ and rank$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 13:59
-	10	@pd<20000531 and ((document with proper with name) and categor$ and rank$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 14:00
-	4	(@pd<20000531 and ((document with proper with name) and categor$ and rank$3)) and cluster	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/05/10 14:00

L Number	Hits	Search Text	DB	Time stamp
1	6004	LIST WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:09
2	10343	SET WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:09
3	13688	(LIST WITH QUER$3) OR (SET WITH QUER$3)	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10
4	2187	PAIR WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10
5	719	((LIST WITH QUER$3) OR (SET WITH QUER$3)) AND (PAIR WITH QUER$3)	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10
6	35204	PAIR WITH FREQUENCY	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10
7	50	(((LIST WITH QUER$3) OR (SET WITH QUER$3)) AND (PAIR WITH QUER$3)) AND (PAIR WITH FREQUENCY)	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10
8	9	((((LIST WITH QUER$3) OR (SET WITH QUER$3)) AND (PAIR WITH QUER$3)) AND (PAIR WITH FREQUENCY)) AND SYNONYM	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:11
9	6	(((((LIST WITH QUER$3) OR (SET WITH QUER$3)) AND (PAIR WITH QUER$3)) AND (PAIR WITH FREQUENCY)) AND SYNONYM) AND @AD<20020201	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:11

	Type	L #	Hits	Search Text	DBs	Time Stamp	Comments	Error Definition	Errors
1	BRS	L1	6004	LIST WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:09			0
2	BRS	L2	10343	SET WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:09			0
3	BRS	L3	13688	1 OR 2	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10			0
4	BRS	L4	2187	PAIR WITH QUER$3	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10			0

	Type	L #	Hits	Search Text	DBs	Time Stamp	Comments	Error Definition	Errors
5	BRS	L5	719	3 AND 4	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10			0
6	BRS	L6	35204	PAIR WITH FREQUENCY	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10			0
7	BRS	L7	50	5 AND 6	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:10			0
8	BRS	L8	9	7 AND SYNONYM	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:11			0

	Type	L #	Hits	Search Text	DBs	Time Stamp	Comments	Error Definition	Errors
9	BRS	L9	6	8 AND @AD<20020201	USPAT; US-PGPUB; EPO; IBM_TDB	2004/05/25 10:11			0

US006006221A

United States Patent [19]

Liddy et al.

[11] Patent Number: 6,006,221

[45] Date of Patent: Dec. 21, 1999

[54] **MULTILINGUAL DOCUMENT RETRIEVAL SYSTEM AND METHOD USING SEMANTIC VECTOR MATCHING**

[75] Inventors: **Elizabeth D. Liddy; Woojin Paik,** both of Syracuse; **Edmund S. Yu,** Dewitt, all of N.Y.; **Ming Li,** Jersey City, N.J.

[73] Assignee: **Syracuse University,** Syracuse, N.Y.

[21] Appl. No.: **08/696,701**

[22] Filed: **Aug. 14, 1996**

Related U.S. Application Data

[60] Provisional application No. 60/002,473, Aug. 16, 1995.

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **707/5**; 704/2; 704/8; 704/9

[58] **Field of Search** 707/3, 4, 5, 6; 704/2, 4, 5, 8, 9

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,148,541	9/1992	Lee et al.	707/2
5,301,109	4/1994	Landauer et al.	704/9
5,418,951	5/1995	Damashek	707/5
5,794,050	8/1998	Dahlgren et al.	395/708

OTHER PUBLICATIONS

Croft Bruce et al., "Applications of Multilingual Text Retrieval," *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, 1996, vol. 5, pp. 98–107.

Liddy, Elizabeth D. et al., "An Overview of DR–LINK and Its Approach to Document Filtering," *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ. Mar. 21–24, 1993, pp. 358–362.

Liddy, Elizabeth D. et al., "Development, Implementation and Testing of a Discourse Model for Newspaper Texts," *ARPA Workshop on Human Language Technology*, Princeton, NJ, Mar. 21–24, 1993, pp. 1–6.

Liddy, Elizabeth D. et al., "DR–LINK's Linguistic–Conceptual Approach to Document Detection," *Proceedings of First Text Retrieval Conference (TREC–1)*, Published Spring 1993, pp. 1–17.

Liddy, Elizabeth D., "DR–LINK: A System Update for TREC–2," *Proceedings of Second Text Retrieval Conference (TREC–2)*, *National Inst. of Standards and Technology*, Aug. 31–Sep. 2, 1993, pp. 1–15.

Liddy, Elizabeth D. et al., "DR–LINK System: Phase I Summary," *Proceedings of the TIPSTER Phase I Final Report*, Sep. 19–23, 1993 (Published 1994).

(List continued on next page.)

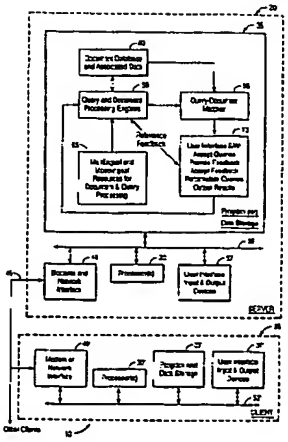
Primary Examiner—Jack M. Choules

Attorney, Agent, or Firm—Townsend and Townsend and Crew LLP

[57] **ABSTRACT**

A document retrieval system where a user can enter a query, including a natural language query, in a desired one of a plurality of supported languages, and retrieve documents from a database that includes documents in at least one other language of the plurality of supported languages. The user need not have any knowledge of the other languages. Each document in the database is subjected to a set of processing steps to generate a language-independent conceptual representation of the subject content of the document. This is normally done before the query is entered. The query is also subjected to a (possibly different) set of processing steps to generate a language-independent conceptual representation of the subject content of the query. The documents and queries can also be subjected to additional analysis to provide additional term-based representations, such as the extraction of information-rich terms and phrases (such as proper nouns). Documents are matched to queries based on the conceptual-level contents of the document and query, and, optionally, on the basis of the term-based representation. The query's representation is then compared to each document's representation to generate a measure of relevance of the document to the query.

35 Claims, 8 Drawing Sheets



OTHER PUBLICATIONS

Liddy, Elizabeth D., "Development and Implementation of a Discourse Model for Newspaper Texts," *Proceedings of the AAAI Symposium on Empirical Methods in Discourse Interpretation and Generation*, Stanford, CA, Dec. 14–17, 1993, pp. 1–6.

Liddy, Elizabeth DuRoss, "An Alternative Representation for Documents and Queries," *Proceedings of the 14th National Online Meeting*, 1993, pp. 279–284.

Liddy, Elizabeth D. et al., "Text Categorization for Multiple Users Based on Semantic Features from a Machine–Readable Dictionary," *ACM Transactions on Information Systems*, vol. 12, No. 3, Jul. 1994, pp. 278–295.

Liddy, Elizabeth D. et al., "Document Retrieval Using Linguistic Knowledge," *Proceedings of the RIAO '94 Conference*, Oct. 11–13, 1994, pp. 106–114.

Liddy, Elizabeth et al., "Detection, Generation, and Expansion of Complex Nominals," *Proceedings of the Workshop on Compound Nouns: Multilingual Aspects of Nominal Composition*, Dec. 2–3, 1994, Geneva, Switzerland, pp. 14–18.

Liddy, Elizabeth D., "Development and Implementation of a Discourse Model for Newspaper Texts," *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ, Mar. 21–24, 1995, pp. 80–84.

Liddy, Elizabeth et al., "A Natural Language Text Retrieval System With Relevance Feedback," *Proceedings of the 16th National Online Meeting*, May 2–6, 1995, pp. 259–261.

Liddy, Elizabeth D., "The Promise of Natural Language Processing for Competitive Intelligence," *Proceedings of 10th International Conference of the Society of Competitive Intelligence Professionals*, May 4–5, 1995, pp. 328–342.

Paik, Woojin et al., "Interpretation of Proper Nouns for Information Retrieval," *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NY, Mar. 21–24, 1993, pp. 1–5.

Paik, Woojin et al., "Categorizing and Standardizing Proper Nouns for Efficient Information Retrieval, " *Corpus Processing for Lexicon Acquisition*, MIT Press, Cambridge, MA, 1995 (Boguraev, B. (ed)) pp. 1–10.

Paik, Woojin, Chronological Information Extraction System (CIES), *Proceedings of the Dagstuhl on Summarizing Text for Intelligent Communication*, Saarbruken, Germany, 1995, pp. 1–5.

Weiner, Michael L. et al., "Intelligent Text Processing, and Intelligence Tradecraft," *The Journal of Association for Global Strategic Intelligence (AGSI)*, Jul. 1995, pp. 1–8.

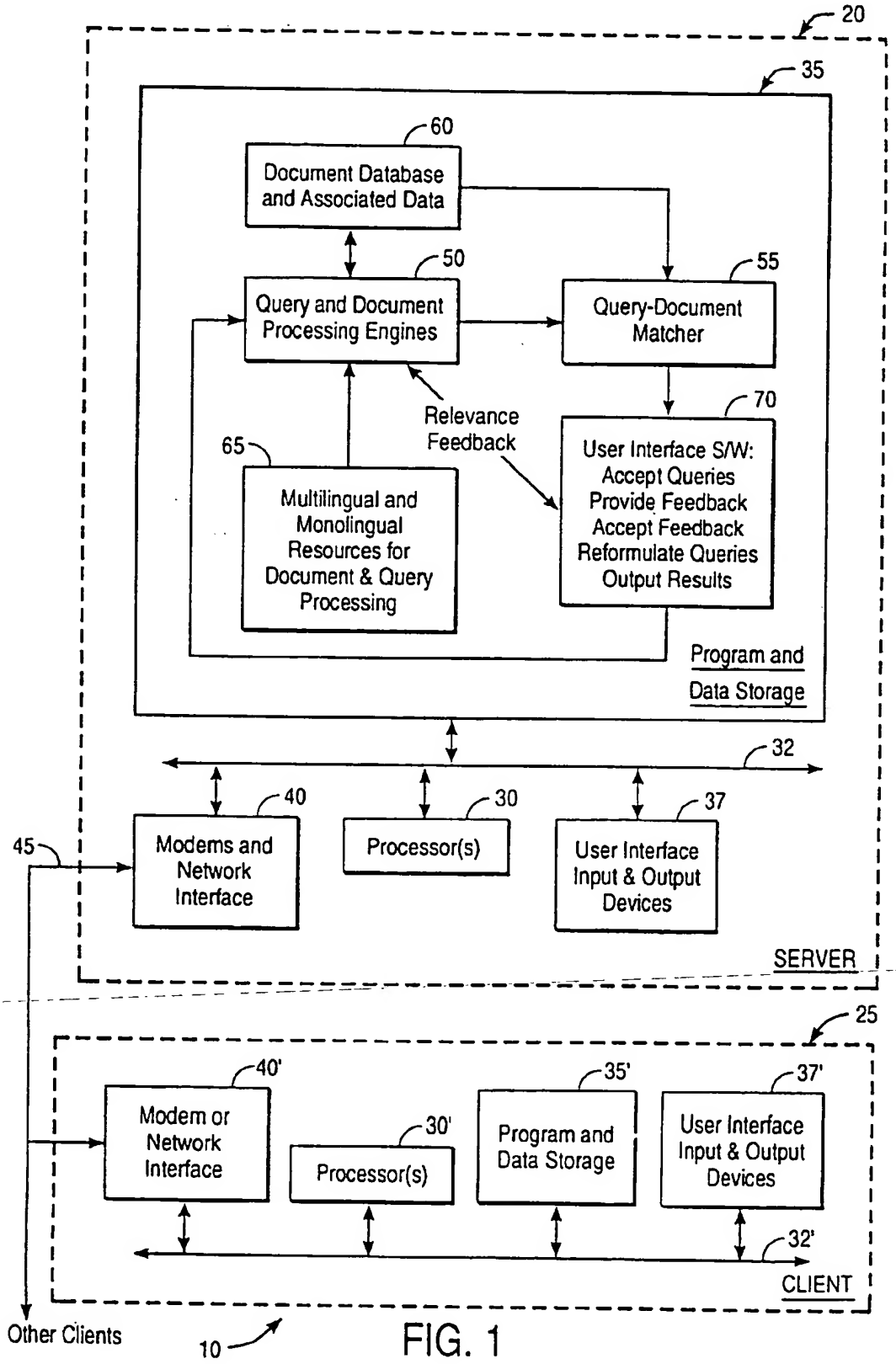

20
35
60
Document Database and Associated Data
50
Query and Document Processing Engines
55
Query-Document Matcher
Relevance Feedback
70
User Interface S/W: Accept Queries Provide Feedback Accept Feedback Reformulate Queries Output Results
65
Multilingual and Monolingual Resources for Document & Query Processing
Program and Data Storage
32
40
30
37
45
Modems and Network Interface
Processor(s)
User Interface Input & Output Devices
SERVER
25
40'
35'
37'
Modem or Network Interface
30'
Processor(s)
Program and Data Storage
User Interface Input & Output Devices
32'
CLIENT
Other Clients
10

FIG. 1

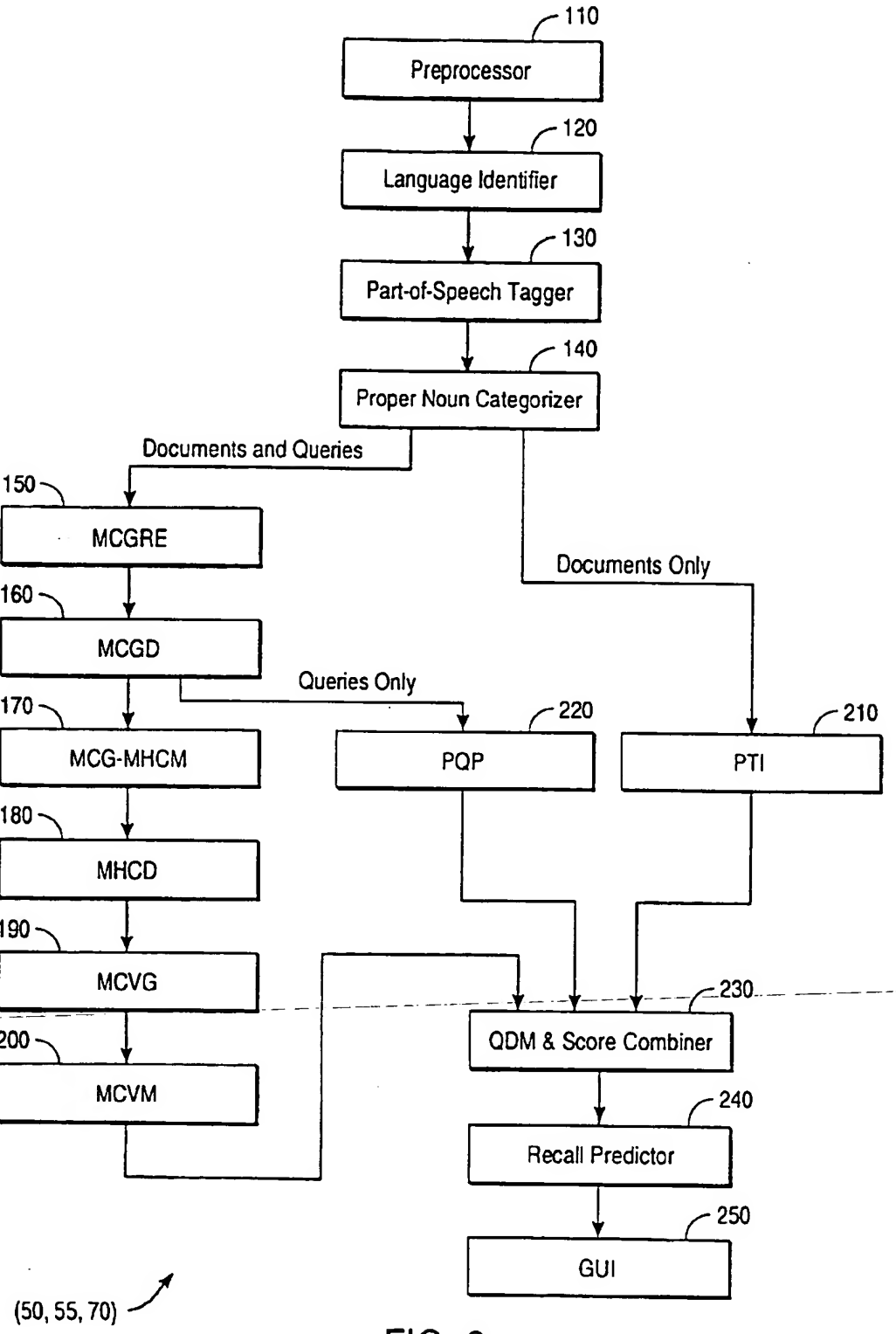

110
Preprocessor
120
Language Identifier
130
Part-of-Speech Tagger
140
Proper Noun Categorizer
Documents and Queries
Documents Only
150
MCGRE
160
MCGD
Queries Only
170
MCG-MHCM
220
PQP
210
PTI
180
MHCD
190
MCVG
230
QDM & Score Combiner
200
MCVM
240
Recall Predictor
250
GUI
(50, 55, 70)

FIG. 2

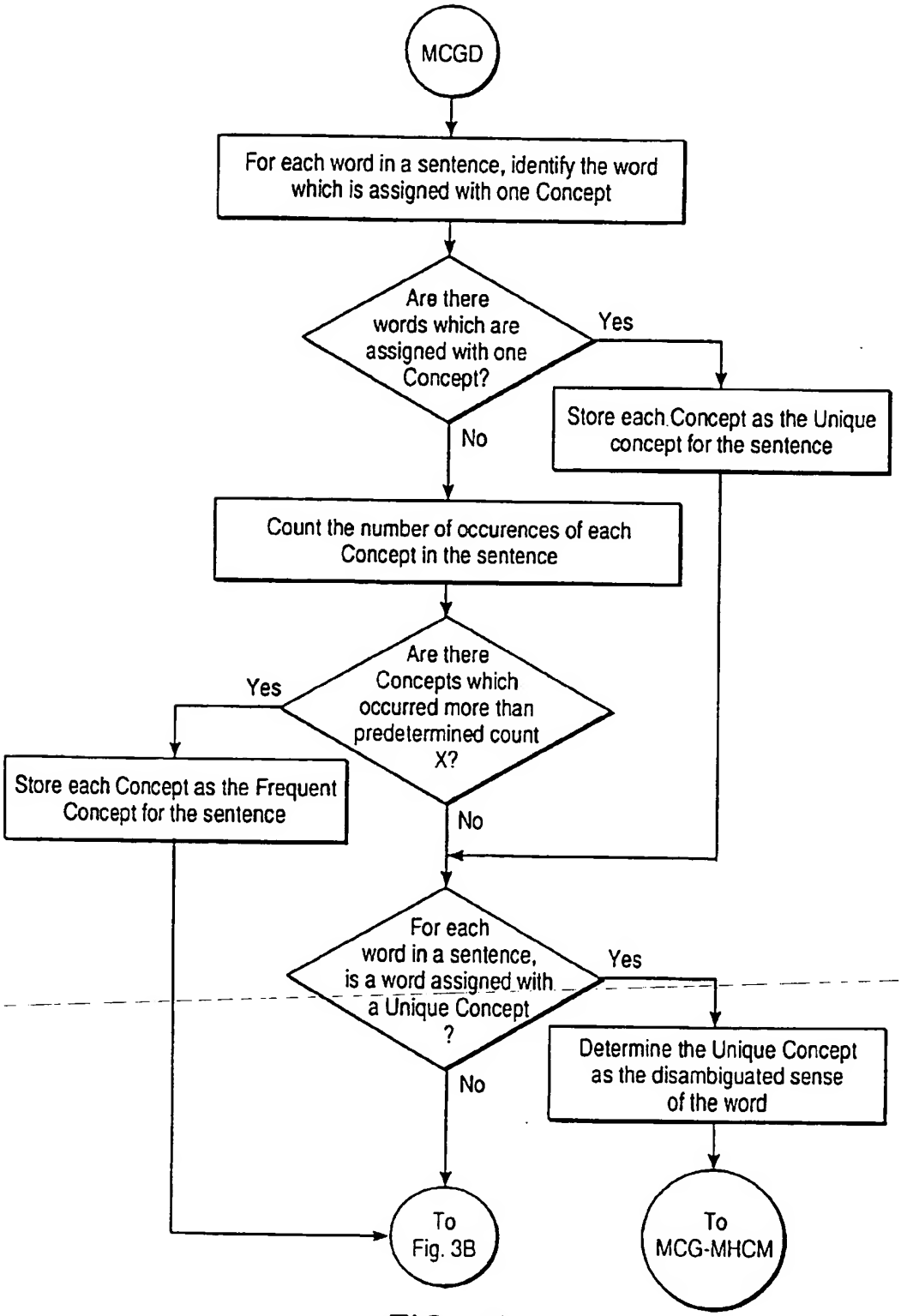

MCGD
For each word in a sentence, identify the word which is assigned with one Concept
Are there words which are assigned with one Concept?
Yes
No
Store each Concept as the Unique concept for the sentence
Count the number of occurences of each Concept in the sentence
Are there Concepts which occurred more than predetermined count X?
Yes
No
Store each Concept as the Frequent Concept for the sentence
For each word in a sentence, is a word assigned with a Unique Concept ?
Yes
No
Determine the Unique Concept as the disambiguated sense of the word
To Fig. 3B
To MCG-MHCM

FIG. 3A

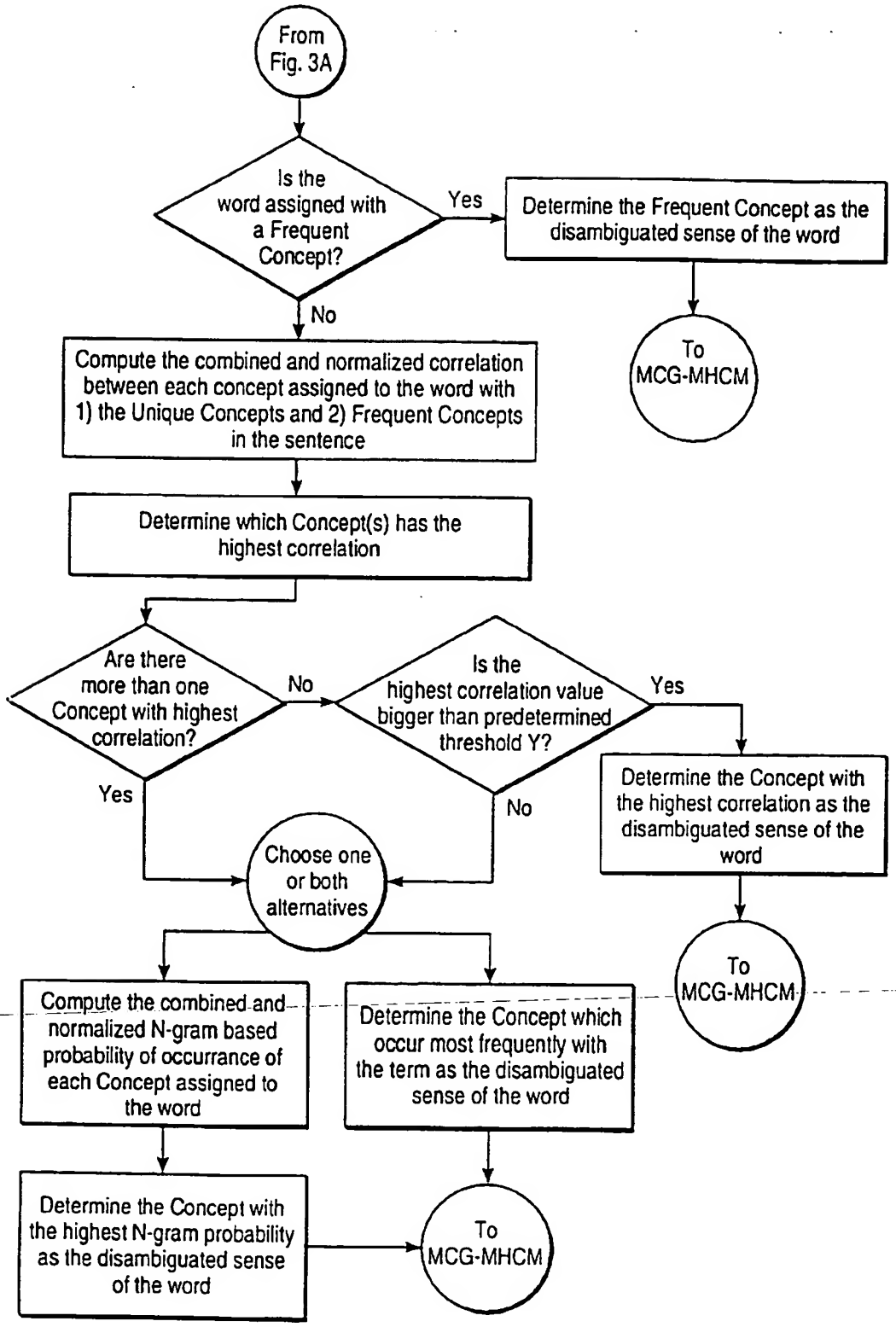

From Fig. 3A
Is the word assigned with a Frequent Concept?
Yes
Determine the Frequent Concept as the disambiguated sense of the word
To MCG-MHCM
No
Compute the combined and normalized correlation between each concept assigned to the word with 1) the Unique Concepts and 2) Frequent Concepts in the sentence
Determine which Concept(s) has the highest correlation
Are there more than one Concept with highest correlation?
No
Is the highest correlation value bigger than predetermined threshold Y?
Yes
Determine the Concept with the highest correlation as the disambiguated sense of the word
To MCG-MHCM
Yes
No
Choose one or both alternatives
Compute the combined and normalized N-gram based probability of occurrance of each Concept assigned to the word
Determine the Concept which occur most frequently with the term as the disambiguated sense of the word
Determine the Concept with the highest N-gram probability as the disambiguated sense of the word
To MCG-MHCM

FIG. 3B

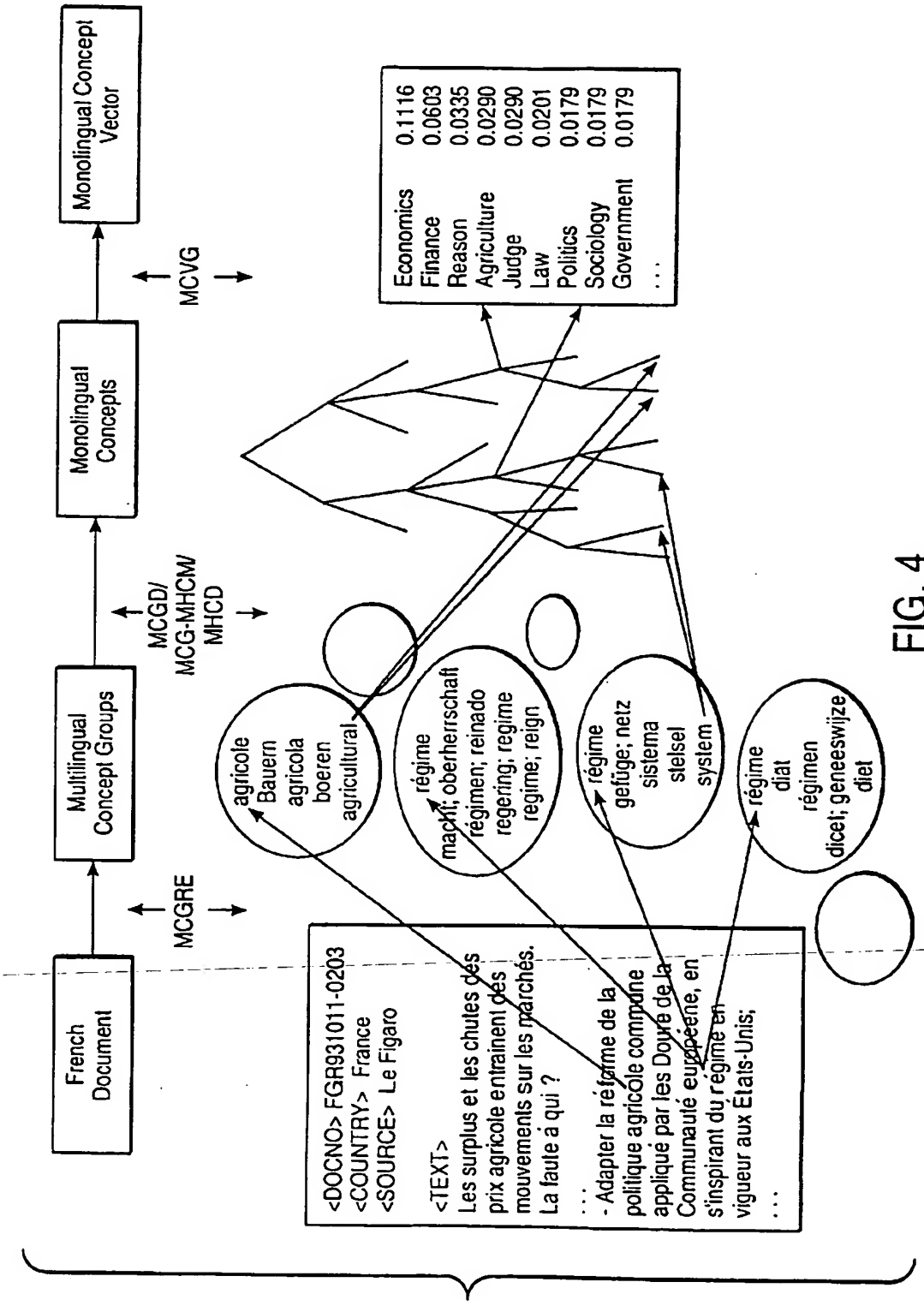

French Document
MCGRE
Multilingual Concept Groups
MCGD/ MCG-MHCM/ MHCD
Monolingual Concepts
MCVG
Monolingual Concept Vector
<DOCNO> FGR931011-0203
<COUNTRY> France
<SOURCE> Le Figaro
<TEXT>
Les surplus et les chutes des prix agricole entrainent des mouvements sur les marchés. La faute á qui ?
...
- Adapter la réforme de la politique agricole commune appliqué par les Doure de la Communauté européene, en s'inspirant du régime en vigueur aux États-Unis;
...
agricole Bauern agricola boeren agricultural
régime machi; oberherrschaft régimen; reinado regering; regime regime; reign
régime gefüge; netz sistema stelsel system
régime diat régimen dicet; geneeswijze diet
Economics 0.1116
Finance 0.0603
Reason 0.0335
Agriculture 0.0290
Judge 0.0290
Law 0.0201
Politics 0.0179
Sociology 0.0179
Government 0.0179
...

FIG. 4

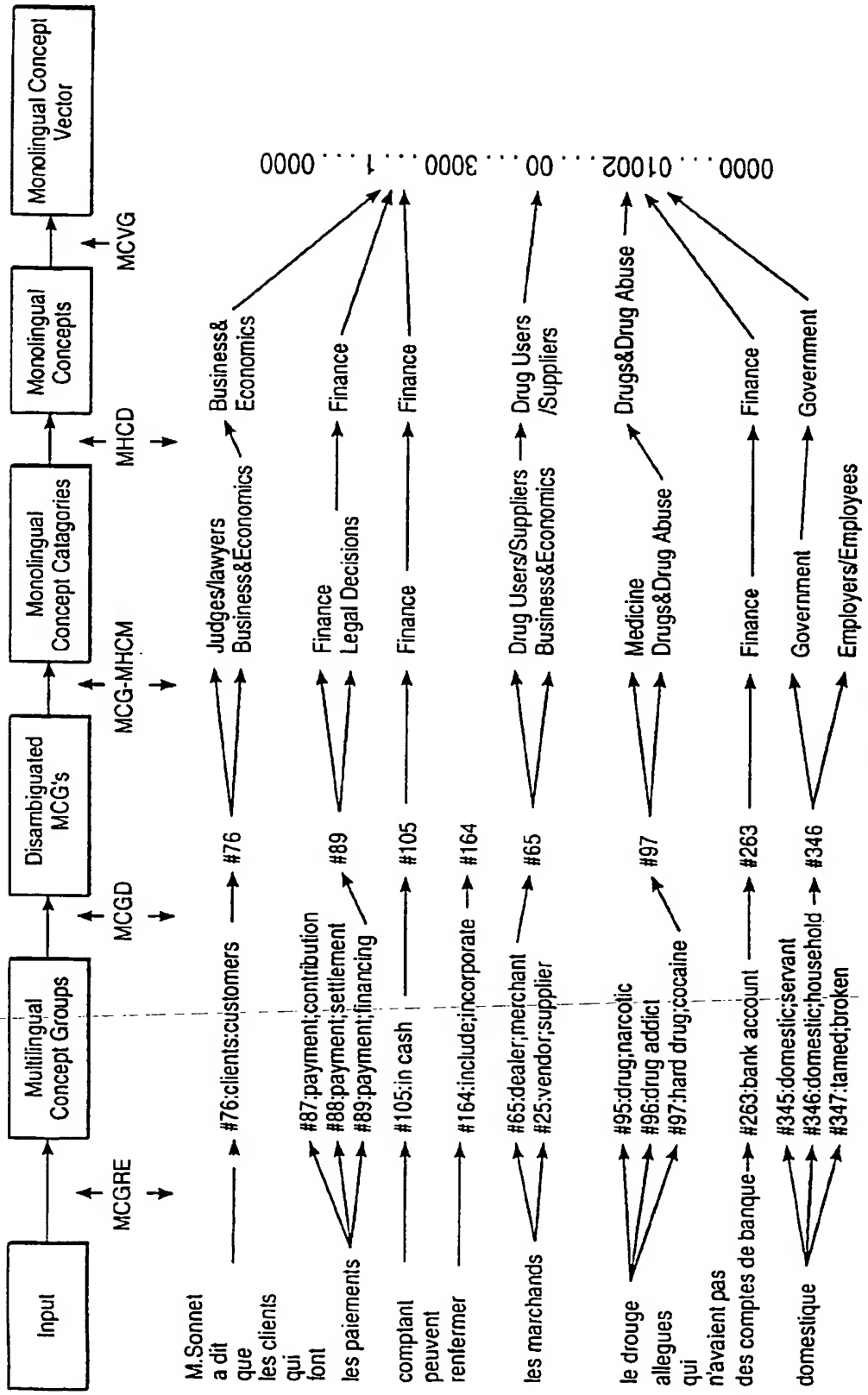

Input
Multilingual Concept Groups
Disambiguated MCG's
Monolingual Concept Catagories
Monolingual Concepts
Monolingual Concept Vector
MCGRE
MCGD
MCG-MHCM
MHCD
MCVG
0000 ... 0100 2 ... 00 ... 3000 ... 1 ... 0000
M.Sonnet a dit que les clients qui font
les paiements
comptant peuvent renfermer
les marchands
le drouge allegues qui n'avaient pas des comptes de banque
domestique
#76:clients:customers
#87:payment;contribution
#88:payment;settlement
#89:payment;financing
#105:in cash
#164:include;incorporate
#65:dealer;merchant
#25:vendor;supplier
#95:drug;narcotic
#96:drug addict
#97:hard drug;cocaine
#263:bank account
#345:domestic;servant
#346:domestic;household
#347:tamed;broken
#76
#89
#105
#164
#65
#97
#263
#346
Judges/lawyers
Business&Economics
Business&Economics
Finance
Legal Decisions
Finance
Finance
Drug Users/Suppliers
Business&Economics
Drug Users /Suppliers
Medicine
Drugs&Drug Abuse
Drugs&Drug Abuse
Finance
Finance
Government
Government
Employers/Employees

FIG. 5

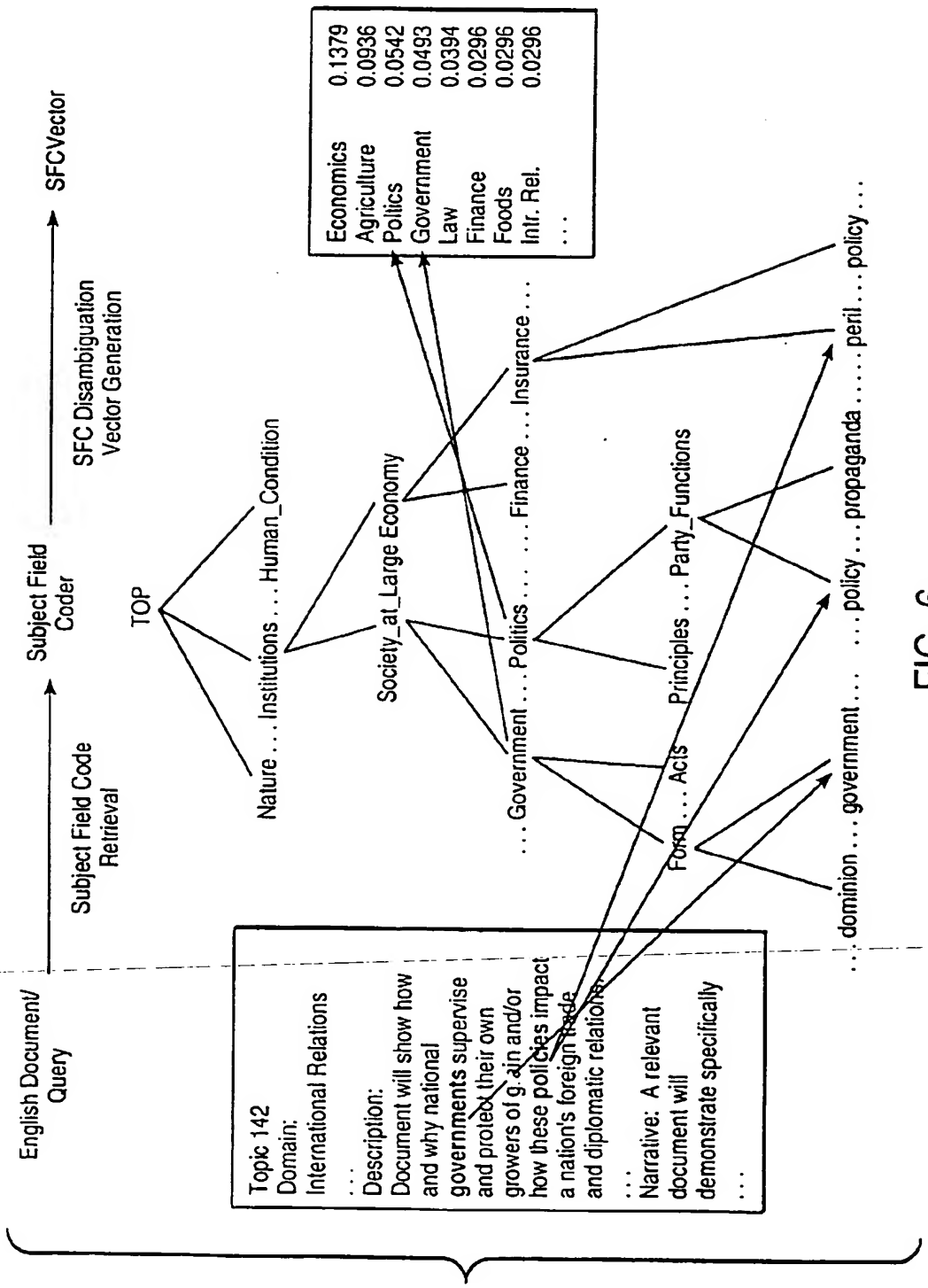

English Document/ Query
Subject Field Code Retrieval
Subject Field Coder
SFC Disambiguation Vector Generation
SFCVector
Topic 142
Domain:
International Relations
...
Description:
Document will show how and why national governments supervise and protect their own growers of g ain and/or how these policies impact a nation's foreign trade and diplomatic relations.
...
Narrative: A relevant document will demonstrate specifically
...
TOP
Nature ... Institutions ... Human_Condition
... Government ... Politics ... ... Finance ... Insurance ...
Society_at_Large Economy
Form ... Acts Principles ... Party_Functions
... dominion ... government ... ... policy ... propaganda ..... peril ... policy ...
Economics 0.1379
Agriculture 0.0936
Politics 0.0542
Government 0.0493
Law 0.0394
Finance 0.0296
Foods 0.0296
Intr. Rel. 0.0296
...

FIG. 6

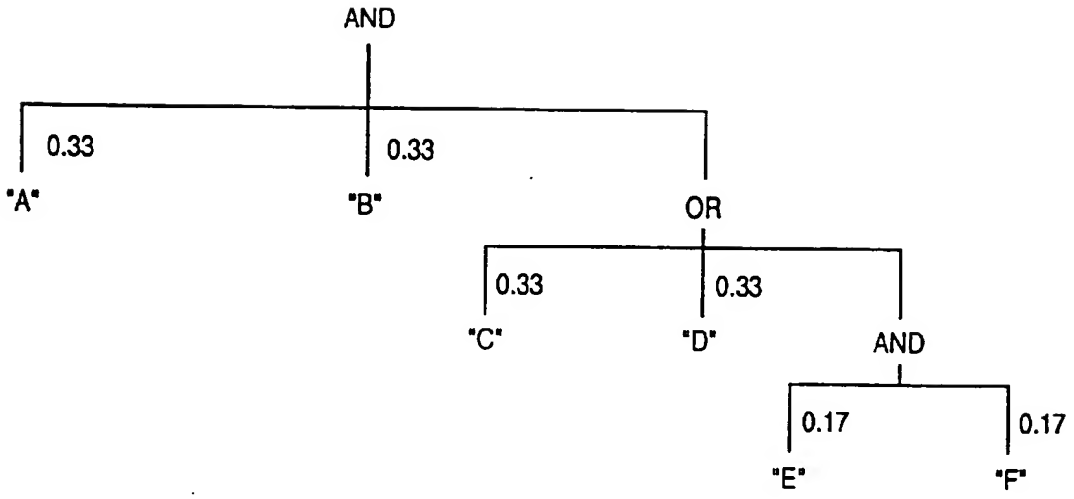


FIG. 7

MULTILINGUAL DOCUMENT RETRIEVAL SYSTEM AND METHOD USING SEMANTIC VECTOR MATCHING

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from, and is a continuation-in-part of, Provisional U.S. patent application Ser. No. 60/002,473, filed Aug. 16, 1995, of Elizabeth D. Liddy entitled FEASIBILITY STUDY OF A MULTILINGUAL TEXT RETRIEVAL SYSTEM, the disclosure of which is hereby incorporated by reference.

The following provisional applications are also hereby incorporated by reference:

1. No. 60/002,451, of Elizabeth D. Liddy, entitled THE PROMISE OF NATURAL LANGUAGE PROCESSING FOR COMPETITIVE INTELLIGENCE;
2. No. 60,002,452, of Elizabeth D. Liddy and Sung H. Myaeng, entitled DR-LINK SYSTEM: PHASE I SUMMARY;
3. No. 60/002,453, of Elizabeth D. Liddy, Edmund S. Yu, Mary McKenna, and Ming Li, entitled DETECTION, GENERATION AND EXPANSION OF COMPLEX NOMINALS;
4. No. 60/002,470, of Elizabeth D. Liddy, Woojin Paik, and Mary McKenna, entitled DEVELOPMENT OF A DISCOURSE MODEL FOR NEWSPAPERS;
5. No. 60/002,471, of Elizabeth D. Liddy, Woojin Paik, Edmund S. Yu, E. S. and Mary McKenna, entitled DOCUMENT RETRIEVAL USING LINGUISTIC KNOWLEDGE; and
6. No. 60/002,472, of Woojin Paik, Elizabeth D. Liddy, Edmund Yu, and Mary McKenna, entitled CATEGORIZING AND STANDARDIZING PROPER NOUNS FOR EFFICIENT INFORMATION RETRIEVAL.

The following applications, including this one, are being filed concurrently, and the disclosure of each other application is incorporated by reference into this application:

U.S. patent application Ser. No. 08/696,701, entitled "MULTILINGUAL DOCUMENT RETRIEVAL SYSTEM AND METHOD USING SEMANTIC VECTOR MATCHING," to Elizabeth D. Liddy, Woojin Paik, Edmund S. Yu, and Ming Li (Attorney Docket No. 17704-2.00);

U.S. patent application Ser. No. 08/698,472, entitled "NATURAL LANGUAGE INFORMATION RETRIEVAL SYSTEM AND METHOD," to Elizabeth D. Liddy, Woojin Paik, Mary McKenna, and Ming Li (Attorney Docket No. 17704-3.00); and

U.S. Pat. application Ser. No. 08/696,702, entitled "USER INTERFACE AND OTHER ENHANCEMENTS FOR NATURAL LANGUAGE INFORMATION RETRIEVAL SYSTEM AND METHOD," to Elizabeth D. Liddy, Woojin Paik, Mary McKenna, Michael Weiner, Edmund S. Yu, Ted Diamond, Bhaskaran Balakrishan, and David Snyder (Attorney Docket No. 17704-4.00).

GOVERNMENT RIGHTS

The U.S. Government has rights in this invention pursuant to Contract No. 94-F159900-000, awarded by the Office of Research and Development, and Contract No. 9331368, awarded by ARPA TRP (U.S. Army Missile Command, Redstone, Ala.).

BACKGROUND OF THE INVENTION

The present invention relates generally to computerized information retrieval, and more specifically to multilingual document retrieval.

A global information economy requires an information utility capable of searching across multiple languages simultaneously and seamlessly. However, when a scientist, patent attorney or patent examiner, student, or any information seeker conducts an electronic search for documents, that search is usually limited to texts in the searcher's native tongue, even though highly relevant information may be freely available in a foreign language. Searching for information across multiple languages invariably proves daunting and expensive, or fruitless and inefficient, and is therefore rarely done.

Patent searching is but one example where limitations of language pose significant obstacles. In prior art terms, all languages are created equal. As a practical matter, a patent examiner in a given country tends to have the most meaningful access to documents in that country's language. Since the most pertinent prior art may be in a different language, patent examiners are often prevented from carrying out an effective examination of patent applications.

The conventional approach to multilingual retrieval is to translate all texts into one common language, then perform monolingual indexing and retrieval. Such systems have several disadvantages. First, the machine translation process, although fully-automated, is often time-consuming and expensive. It is also highly inefficient, since all documents must be translated even though only a small fraction of documents will be relevant to any given query.

Second, the process of translation inevitably introduces errors and ambiguities into the translated document, making subsequent indexing and retrieval troublesome. For example, translation systems perform poorly with specialized discourse (medicine, law, etc.), and are often unable to disambiguate polysemous words (those words with multiple meanings) correctly.

SUMMARY OF THE INVENTION

The present invention provides document retrieval techniques that enable a user to enter a query, including a natural language query, in a desired one of a plurality of supported languages, and retrieve documents from a database that includes documents in at least one other language of the plurality of supported languages. The user need not have any knowledge of the other languages. The present invention thus makes simultaneously searching multiple languages viable and affordable. Even if the documents of interest are all in one language, the invention gives a user whose native language is different the ability to enter queries in the user's native language.

In short, each document in the database is subjected to a set of processing steps to generate a language-independent conceptual representation of the subject content of the document. This is normally done before the query is entered. The query is also subjected to a (possibly different) set of processing steps to generate a language-independent conceptual representation of the subject content of the query. The documents and queries can also be subjected to additional analysis to provide additional term-based representations, such as the extraction of information-rich terms and phrases (such as proper nouns).

Documents are matched to queries based on the conceptual-level contents of the document and query, and,

optionally, on the basis of the term-based representation. For example, the matching can be based in part on the co-occurrence of information-rich terms and phrases, or appropriate expansions or synonyms.

The query's representation is then compared to each document's representation to generate a measure of relevance of the document to the query. Results can be browsed using a graphical interface, and individual documents (or document clusters) that seem highly relevant can be used to inform subsequent queries for relevance feedback. The system may also perform a surface-level, gloss transliteration of the foreign text, sufficient enough for a non-fluent reader to gain a basic understanding of the document's contents.

In specific embodiments, the language-independent conceptual representation of the subject content of the document, and that of the query, is a fixed-length vector based on a set of subject content categories and subcategories. A current implementation supports English, French, German, Spanish, Dutch, and Italian. However, the system is modular, and as additional languages are added to the document databases, those languages become searchable.

The invention, by abstracting the documents and queries into language-independent conceptual form, avoids the need for machine translation of the query or the database of documents. Only those documents which appear highly relevant to the searcher need be considered as candidates for translation (human or machine).

A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a multilingual information retrieval system embodying the present invention;

FIG. **2** is a block diagram of the text processing portion of the system;

FIGS. **3**A and **3**B, taken together, provide a flowchart showing the operation of the multilingual concept group disambiguator (MCGD);

FIG. **4** is a high-level diagram showing the processing of French input text to a monolingual concept vector;

FIG. **5** is a more detailed diagram showing the two stages of disambiguation in the processing of French input text to a monolingual concept vector;

FIG. **6** shows an example of a portion of the processing in a monolingual system; and

FIG. **7** shows a logical tree representation of an exemplary query.

DESCRIPTION OF SPECIFIC EMBODIMENTS

1.0 Introduction

The present invention is embodied in a multilingual document retrieval system, **10**, sometimes referred to as CINDOR (Conceptual INterlingua DOcument Retrieval). The CINDOR system is capable of accepting a user's query stated in any one of a plurality of supported languages while seamlessly searching, retrieving and relevance-ranking documents written in any of the supported languages. The system further offers a "gloss" transliteration of target documents, once retrieved, sufficient for a surface understanding of the document's contents.

Unless otherwise stated, the term "document" should be taken to mean text, a unit of which is selected for analysis, and to include an entire document, or any portion thereof, such as a title, an abstract, or one or more clauses, sentences, or paragraphs. A document will typically be a member of a document database, referred to as a corpus, containing a large number of documents. Such a corpus can contain documents in any or all of the plurality of supported languages.

Unless otherwise stated, the term "query" should be taken to mean text that is input for the purpose of selecting a subset of documents from a document database. While most queries entered by a user tend to be short compared to most documents stored in the database, this should not be assumed. The present invention is designed to allow natural language queries.

Unless otherwise stated, the term "word" should be taken to include single words, compound words, phrases, and other multi-word constructs. Furthermore, the terms "word" and "term" are often used interchangeably. Terms and words include, for example, nouns, proper nouns, complex nominals, noun phrases, verbs, adverbs, numeric expressions, and adjectives. This includes stemmed and non-stemmed forms.

The disclosures of all articles and references, including patent documents, mentioned in this application are incorporated herein by reference as if set out in full.

2.0 System Hardware Overview

FIG. **1** is a simplified block diagram of a computer system **10** embodying the multilingual text retrieval system of the present invention. The invention is typically implemented in a client-server configuration including a server **20** and numerous clients, one of which is shown at **25**. The use of the term "server" is used in the context of the invention, where the server receives queries from (typically remote) clients, does substantially all the processing necessary to formulate responses to the queries, and provides these responses to the clients. However, server **20** may itself act in the capacity of a client when it accesses remote databases located on a database server. Furthermore, while a client-server configuration is shown, the invention may be implemented as a standalone facility, in which case client **25** would be absent from the figure.

The hardware configurations are in general standard, and will be described only briefly. In accordance with known practice, server **20** includes one or more processors **30** that communicate with a number of peripheral devices via a bus subsystem **32**. These peripheral devices typically include a storage subsystem **35** (memory subsystem and file storage subsystem), a set of user interface input and output devices **37**, and an interface to outside networks, including the public switched-telephone network. This interface is shown schematically as a "Modems and Network Interface" block **40**, and is coupled to corresponding interface devices in client computers via a network connection **45**.

Client **25** has the same general configuration, although typically with less storage and processing capability. Thus, while the client computer could be a terminal or a low-end personal computer, the server computer would generally need to be a high-end workstation or mainframe. Corresponding elements and subsystems in the client computer are shown with corresponding, but primed, reference numerals.

The user interface input devices typically includes a keyboard and may further include a pointing device and a scanner. The pointing device may be an indirect pointing device such as a mouse, trackball, touchpad, or graphics tablet, or a direct pointing device such as a touchscreen incorporated into the display. Other types of user interface input devices, such as voice recognition systems, are also possible.

The user interface output devices typically include a printer and a display subsystem, which includes a display controller and a display device coupled to the controller. The display device may be a cathode ray tube (CRT), a flat-panel device such as a liquid crystal display (LCD), or a projection device. Display controller provides control signals to the display device and normally includes a display memory for storing the pixels that appear on the display device. The display subsystem may also provide non-visual display such as audio output.

The memory subsystem typically includes a number of memories including a main random access memory (RAM) for storage of instructions and data during program execution and a read only memory (ROM) in which fixed instructions are stored. In the case of Macintosh-compatible personal computers the ROM would include portions of the operating system; in the case of IBM-compatible personal computers, this would include the BIOS (basic input/output system).

The file storage subsystem provides persistent (non-volatile) storage for program and data files, and typically includes at least one hard disk drive and at least one floppy disk drive (with associated removable media). There may also be other devices such as a CD-ROM drive and optical drives (all with their associate removable media). Additionally, the system may include drives of the type with removable media cartridges. The removable media cartridges may, for example be hard disk cartridges, such as those marketed by Syquest and others, and flexible disk cartridges, such as those marketed by Iomega. As noted above, one or more of the drives may be located at a remote location, such as in a server on a local area network or at a site on the Internet's World Wide Web.

In this context, the term "bus subsystem" is used generically so as to include any mechanism for letting the various components and subsystems communicate with each other as intended. With the exception of the input devices and the display, the other components need not be at the same physical location. Thus, for example, portions of the file storage system could be connected via various local-area or wide-area network media, including telephone lines. Similarly, the input devices and display need not be at the same location as the processor, although it is anticipated that the present invention will most often be implemented in the context of PCs and workstations.

Bus subsystem **32** is shown schematically as a single bus, but a typical system has a number of buses such as a local bus and one or more expansion buses (e.g., ADB, SCSI, ISA, EISA, MCA, NuBus, or PCI), as well as serial and parallel ports. Network connections are usually established through a device such as a network adapter on one of these expansion buses or a modem on a serial port. The client computer may be a desktop system or a portable system.

The user interacts with the system using user interface devices **37'** (or devices **37** in a standalone system). For example, client queries are typically entered via a keyboard, communicated to client processor **30'**, and thence to modem or network interface **40'** over bus subsystem **32'**. The query is then communicated to server **20** via network connection **45**. Similarly, results of the query are communicated from the server to the client via network connection **45** for output on one of devices **37'** (say a display or a printer), or may be stored on storage subsystem **35'**.

3.0 Text Processing (Software) Overview

3.1 Basic Functionality

The server's storage subsystem **35** shows the basic programming and data constructs that provide the functionality of the CINDOR system. The CINDOR software is designed to (1) process text stored in digital form or entered in digital form on a computer terminal to create a database file recording the manifold contents of the text, and (2) match discrete texts (documents) to the requirements of a user's query text. CINDOR provides rich, deep processing of text by representing and matching documents and queries at the lexical, syntactic, semantic and discourse levels, not simply by detecting the co-occurrence of words or phrases. A user of the system is able to enter queries, in the user's own language, as fully-formed sentences, with no requirement for special coding, annotation or the use of logical operators.

The system is modular and performs staged processing of documents, with each module adding a meaningful annotation to the text. For matching, a query undergoes analogous processing to determine the requirements for document matching. The system generates both conceptual and term-based alternative representations of the documents and queries.

The server's storage subsystem **35**, as shown in FIG. **1**, contains the basic programming and data constructs that provide the functionality of the CINDOR system. The processing modules include a set of processing engines, shown collectively in a processing engine block **50**, and a query-document matcher **55**. It should be understood, however, that by the time a user is entering queries into the system, the relevant document databases will have been processed and annotated, and various data files and data constructs will have been established. These are shown schematically as a "Document Database and Associated Data" block **60**, referred to collectively below as the document database. An additional set of resources **65**, possibly including some derived from the corpus at large, is used by the processing engines in connection with processing the documents and queries. As will be described below, resources **65** include a number of multilingual resources.

User interface software **70** allows the user to interact with the system. The user interface software is responsible for accepting queries, which it provides to processing engines **50**. The user interface software also provides feedback to the user regarding the query, and, in specific embodiments accepts responsive feedback from the user in order to reformulate the query. The user interface software also presents the results of the query to the user and reformats the output in response to user input. User interface software **70** is preferably implemented as a graphical user interface (GUI), and will often be referred to as the GUI.

Processing of documents and queries follows a modular progression, with documents being matched to queries based on matching (1) their conceptual-level contents, and (2) various term-based and logic representations such as the frequency/co-occurrence of proper nouns. At the conceptual level of matching, each substantive word in a document or query is assigned a concept category, and these category frequencies are summed to produce a vector representation of the whole text. Proper nouns are considered separately and, using a modified, fuzzy Boolean representation, matching occurs based on the frequency and co-occurrence of proper nouns in documents and queries. The principles applied to the proper noun matching are applicable to matching for other terms and parts of speech, such as complex nominals (CNs) and single terms.

While FIG. **1** shows documents and queries being processed, it should be understood that the documents would normally have been processed during an initial phase of setting up the document database and related structures, with relevant information extracted from the documents and

indexed as part of the database. Accordingly, in the discussion that follows, when reference is made to documents and queries being processed in a particular way, it is generally to be understood that the processing of documents and queries would be occurring at different times.

3.2 Processing Module Overview

FIG. **2** is a block diagram showing the set of modules that form processing engines **50**, query-document matcher **55**, and user interface software **70**. Documents and queries are processed by this set of modules that provide a language-independent conceptual representation of each document and query. (As mentioned above, the documents and queries are also subjected to separate processing.) In this context, the modifier "language-independent" means that the documents and queries are all abstracted to a set of categories expressed in a common representation without regard to their original language. This processing is distinct from machine translation, as will be seen below. This does not mean, however, that retrieved documents could not then be translated, by machine or otherwise, if deemed appropriate by the user.

The set of modules that perform the processing to generate the conceptual representation and the term-based representation includes:

- a preprocessor **110**,
- a language identifier (LI) **120**,
- a part of speech (POS) tagger **130**,
- a proper noun categorizer (PNC) **140**,
- a multilingual concept group retrieval engine (MCGRE) **150**,
- a multilingual concept group disambiguator (MCGD) **160**,
- a multilingual concept group to monolingual hierarchical concept mapper (MCG-MHCM) **170**,
- a monolingual hierarchical concept category disambiguator (MHCD) **180**,
- a monolingual category vector generator (MCVG) **190**,
- a monolingual category vector matcher (MCVM) **200**,
- a probabilistic term indexer (PTI) **210**,
- a probabilistic query processor (PQP) **220**,
- a query to document matcher (QDM) and score combiner **230**,
- a recall predictor **240**, and
- a graphical user interface (GUI) **250**.

The output of MCVG **190** is a monolingual category vector (also referred to as the semantic vector, or simply the vector) for each document and query, and represents the documents or query at a language-independent conceptual level. The query's monolingual category vector is matched or compared with monolingual category vectors of the documents by MCVM **200**. The output from MCVM **200** provides a measure of relevance (score) for each document with respect to the query.

While this information alone could be used to rank documents, it is preferred to subject the documents and the queries to an additional set of operations to provide additional bases for evaluating relevance. To this end, the document information output from PNC **140** is communicated to PTI **210**, while the query information from MCGD **160** is communicated to PQP **220**. PTI **210** and PQP **220** provide term-based representations of the documents and query, respectively.

The outputs from MCVM **200**, PTI **210**, and PQP **220** are evaluated by QDM and score combiner **230**, which provides a score for each document. The output scores are processed by recall predictor **240** so as to select a proper set for output. The results are stored, and typically presented to a user for browsing at GUI **250**.

The processing modules can be grouped at a higher level. Preprocessor **110**, LI **120**, POS tagger **130**, and PNC **140** perform initial processing for tagging and identification; MCGRE **150**, MCGD **160**, MCG-MHCM **170**, MHCD **180**, and MCVG **190** generate conceptual-level representations of the documents and queries; PTI **210** and PQP **220** generate term-based representations of the documents and queries; MCVM **200** and QDM and score combiner **230** correlate the document and query information to provide an evaluation of the documents; and recall predictor **240** and GUI **250** are concerned with presenting the results to the user.

A number of the processing modules mentioned above rely on associated resources, including databases and the like. While these resources will be described in connection with the following detailed descriptions of the modules, they are enumerated here for clarity.

PNC **140**:

proper noun knowledge databases (PKND).

MCGRE **150**:

multilingual concept database (MCD).

MCGD **160**:

multilingual concept group n-gram probability database,

multilingual concept group correlation matrix (MCGCM), and

frequency database.

MCG-MHCM **170**:

monolingual hierarchical concept dictionary (MHCD).

MHCD **180**:

monolingual category correlation matrix (MCCM).

MCVG **190**:

index.

What follows is a module by module description of the system.

4.0 Initial Processing and Tagging

4.1 Preprocessor **110**

Preprocessor **110** accepts raw, unformatted text and transfers this to a standard format suitable for further processing by CINDOR. The preprocessor performs document-level processing as follows:

The beginning and end of documents are identified and marked.

Discourse-Level tagging occurs. Various fields and text types are identified and tagged in a document, including "headline," "sub-text headline," "date," and "caption."

All text is annotated with SGML-like tags (standard generalized markup language, set forth as ISO standard 8879).

4.2 Language Identifier (LI) **120**

LI **120** determines by means of a combination of n-gram and word frequency analysis the language of the input document. The output of the LI is the document plus its language identification tag.

Two parallel approaches for language identification are employed. The first approach operates by scanning documents for a distribution of language-discriminant, common single words. The occurrence, frequency and distribution of these words in a document is compared against the same distributions gathered from a representative corpus of documents in each of the supported languages. The second approach involves locating common word/character sequences unique to each language. Such sequences may

form actual words that often occur, such as conjunctions, or a mix of words, punctuation and character strings. Language identification involves scanning each document until a target character sequence is located.

It should be realized that the LI is not necessary if the documents are already tagged as to their language.

4.3 Part of Speech (POS) Tagger **130**

The language dependent, probabilistic, POS tagger **130** determines the appropriate part of speech for each input word in the document and outputs a part of speech tagged document, plus its language identification tag.

POS tagger **130** is used to identify various substantive words such as nouns, verbs, adjectives, proper nouns, and adverbs in each of the supported languages. Various functional words such as conjuncts are tagged as stop-words and are not used for matching purposes. Each language-specific POS tagger is a commercial off-the-shelf (COTS) technology.

4.4 Proper Noun Identifier & Categorizer (PNC) **140**

In addition to the parts of speech processing in POS tagger **130**, additional processing of proper nouns occurs in a separate processing module, namely PNC **140**, which performs the following tasks:

Identifies and tags adjacent proper nouns in a text using the Proper Noun Boundary Identifier (PNBI). The PNBI uses various heuristics developed through multilingual corpus analysis to bracket adjacent proper nouns (e.g., IBM Corporation) and bracket proper nouns with embedded conjunctions and prepositions (e.g., the Bill of Rights). For example, one heuristic takes the form of a database of proper nouns such as University or Mayor that are frequently linked to proximate proper nouns by the preposition "of." In another scheme, specific instantiations of adjacent proper nouns can be stored in a database. Each supported language has an independent array of tools and embedded databases for detecting and tagging adjacent proper nouns.

Normalizes each proper noun to its standard form. For example, "IBM" and the colloquial "Big Blue" are both normalized to the standard form of "International Business Machines, Inc." in the knowledge database.

Expands group proper nouns to their constituent members using the proper noun knowledge databases. For example, the group proper noun "European Community" is expanded to all member countries (Great Britain, France, Germany, etc.). Later matching would consider all expansions on the original proper noun group.

Assigns monolingual concept-level categories from a proper noun hierarchical classification scheme to certain proper nouns or portions of proper nouns. The proper noun classification scheme is based on algorithmic machine-aided corpus analysis in each supported language. In a specific implementation, the classification is hierarchical, consisting of nine branch nodes and thirty terminal nodes. Clearly, this particular hierarchical arrangement of codes is but one of many arrangements that would be suitable. Table 1 shows a representative set of proper noun concept categories and subcategories.

TABLE 1

Proper Noun Categories and Subcategories	
Geographic Entity:	Human:
City	Person
Port	Title
Airport	Document:

TABLE 1-continued

Proper Noun Categories and Subcategories	
Island	Document
County	Equipment:
Province	Software
Country	Hardware
Continent	Machines
Region	Scientific:
Water	Disease
Geographic Miscellaneous	Drugs
Affiliation:	Chemicals
Religion	Temporal:
Nationality	Date
Organization:	Time
Company	Miscellaneous:
Company Type	Miscellaneous
Government	
U.S. Government	
Organization	

Classification is accomplished by reference to an array of knowledge bases and context heuristics, which collectively define the proper noun knowledge database (PNKD). The PNKD was built by analyzing a large corpus of texts, and contains the following different types of information which are used to categorize and standardize proper nouns in texts:

(1) lists of common prefixes and suffixes which suggest certain types of proper noun categories;

(2) lists of contextual linguistic clues which suggest certain types of proper noun categories;

(3) lists of commonly used alternative names of the highly frequent proper nouns; and

(4) lists of highly common proper nouns and the categories to which the proper nouns belong.

Classification includes (but is not limited to) company name, organization names, geographic entities, government units, government and political officials, patented and trademarked products, and social institutions. Monolingual proper noun concept categories are used to help form the monolingual category vector representation of both the document and query (see later descriptions). As noted above, the documents and queries output from PNC **140** are communicated to MCGRE **150**, while in the specific implementation the documents only are communicated to PTI **210**.

5.0 Generation of Conceptual Level Representation

5.1 Multilingual Concept Group Retrieval Engine (MCGRE) **150**

Modules **150** through **190** (i.e., MCGRE **150**, MCGD **160**, MCG-MHCM **170**, MHCD **180**, and MCVG **190**) are used to generate monolingual category vector codes of the subject-contents of both documents and queries. This process involves recognizing various information-rich words or parts of speech in a native language text, assigning a single code to these words or phrases that establishes its conceptual-level meaning, then mapping this conceptual-level representation to an English language, hierarchical system of concept codes for vector creation.

The first of these modules, MCGRE **150**, accepts the language-identified, part-of-speech tagged, input text and retrieves from the multilingual concept database any and all of the concept groups to which each input word belongs. Polysemous words (those words with multiple meanings)

will have multiple concept group assignments at this stage. The output of the MCGRE **150**, when run over a document, will be sentence-delimited strings of words, each word or phrase of which has been tagged with the codes of all the multilingual concept groups to which various senses of the word/phrase belongs.

This process incorporates:

(a) Deinflection of words (finding their root form);

(b) Locating clitics (articles or pronouns attached to words or punctuation, as with the French "l'enfant");

(c) Identifying and splitting compound words (words consisting of two or more linked words); and

(d) Mapping each word to all possible corresponding concept categories using the multilingual concept database (MCD).

The MCD is a language-independent knowledge database comprising a collection of non-hierarchical concept groups. There are about 10,000 concept groups in a current implementation. Within each concept group is a collection of words or phrases, in multiple languages, that are conceptually synonymous or near-synonymous. Usually all members of a given concept group belong to the same part of speech. It is possible that many words in a given language will occur in a given concept group, or that a given word or phrase will occur in multiple concept groups. The number of concept groups that a given word or phrase occupies is dependent on the degree of polysemy of that word or phrase. For example, a word that has three possible senses may occupy three different concept groups. Each group is considered a language-independent concept. Note that the MCD differs from a thesaurus because the concept groups are not linked by broader or narrower relations. The MCD differs from a dictionary translation because the MCD grouping is by synonymous words, not by translation definition.

5.2 Multilingual Concept Group Disambiguator (MCGD) **160**

The input to MCGD **160** is the fully-tagged text stream from MCGRE **150** with polysemous words having multiple concept-category tags. The function of the MCGD is to select the single most appropriate concept group from the multilingual concept database for all those input words for which multiple concept group tags have been retrieved. The output of the MCGD is a fully-tagged text stream with a single multilingual concept group for each word in the input text. The processing performed by this module is similar to that discussed in copending commonly-owned U.S. patent application Ser. No. 08/135,815, filed Oct. 12, 1993, entitled "Natural Language Processing System For Semantic Vector Representation Which Accounts For Lexical Ambiguity," to Elizabeth D. Liddy, Woojin Paik, and Edmund Szu-Li Yu, though modified for a multilingual system. The application mentioned immediately above, hereinafter referred to as "Natural Language Processing," is hereby incorporated by reference for all purposes.

FIGS. **3A** and **3B**, taken together, provide a flowchart showing the operation of MCGD **160**. MCGD **160** processes text a sentence at a time, using the original language of the input text as a useful context for selecting the most appropriate sense of the words in a sentence.

If disambiguation is needed (the input word belongs to more than one concept group), then the MCGD will select the appropriate concept group using three sources of linguistic evidence. These are: (a) Local Context, (b) Domain Knowledge, and (c) Global Information, which are used as follows.

5.2.1 Local Context

If a word in the sentence has been tagged with only one concept group code, this concept group code is considered Unique. Further, if there are any concept group codes which have been assigned to more than a predetermined number of words within the sentence being processed, these concept group codes are considered Frequentcodes. These two types of locally determined concept group codes are used as "anchors" in the sentence for disambiguating the remaining words. If any of the ambiguous (polysemous) words in the sentence have either a Unique or Frequent concept group code amongst their codes, that concept group code is selected and that word is thereby disambiguated.

FIG. **3A** shows this process where MCGD **160** determines whether a given multilingual concept group code is Unique or Frequent, and further whether a given ambiguous word has a Unique or Frequent code as one of its assigned codes. To the extent that the word is associated with a Unique or Frequent code, that Unique or Frequent code is used.

However, a word which has no overlap between its concept group codes and the Unique or Frequent concept group codes for that sentence cannot be disambiguated using local context evidence, and must be evaluated by the next source of linguistic evidence, Domain Knowledge.

5.2.2 Domain Knowledge

Domain Knowledge representations reflect the extent to which words of one concept group tend to co-occur with words of the other concept groups (hence the notion of the domain predicting the sense). For each word which has not had one of its multiple concept group codes selected using local information, the system consults the multilingual concept group correlation matrix (MCGCM) to select an appropriate concept group code from the multiple concept group codes attached to the input word.

The MCGCM is an optional knowledge database that reflects observed document level co-occurrence patterns across a large corpus of single language documents. This correlation matrix is built from the training data to be used as an additional knowledge source to disambiguate multiple concept groups which are assigned to the terms in both query and documents. The training data which is used to construct the correlation matrix is either all possible concept groups assigned to each term in the texts, or the partially disambiguated concept groups in the texts. Thus, the construction of the correlation matrix does not require manual intervention.

This correlation matrix is constructed from the correlation information among all concept groups assigned to terms in one document. The collection of the correlation information is summed and normalized to get the stable correlation among all possible concept groups (i.e., each concept group will have a correlation value against all the other possible concept groups.)

The MCGCM consists of unweighted Pearson's product-moment correlation coefficients for all of the multilingual database concept group pairs using within-document occurrences as the unit of analysis. The result will be correlation scores for each concept group pair between −1 and +1. Within a sentence a word with multiple concepts categories is disambiguated to the single concept category that is most highly correlated with the Unique or Frequent concept category. If several Unique or Frequent anchor words exist, the ambiguous word is disambiguated to the correct category of the anchor word with the highest overall correlation coefficient.

The Local and Domain Knowledge evidence sources can select a concept group code for each word in the sentence, if at least a single Unique or Frequent concept group code was selected as an "anchor" code for the sentence. But, for words in those sentences for which an "anchor" was not

found, the third evidence source, Global Knowledge, will need to be consulted.

5.2.3 Global Knowledge

Global Knowledge simulates the observation made in human sense disambiguation that more frequently used senses of words are cognitively activated in preference to less frequently used senses of words. Therefore, the words not yet disambiguated by Local Context or Domain Knowledge will now have their multiple concept group codes compared to a Global Knowledge database source, referred to as the frequency database. The database is an external, off-line sense-tagging of parallel corpora with the correct concept group code for each word. The disambiguated parallel corpora will provide frequencies of each word's usage as a particular sense (equatable to concept group) in the sample corpora. The most frequent sense is selected as the concept category.

The frequency database can be constructed in any of the following three ways:

(1) Collect the most frequent sense information from partially or fully sense-disambiguated texts (the training data to collect sense frequency information can be built either manually or automatically). Training data can be built automatically from the output from MCGD module without the frequency database OR the output from automatic sense comparison using multilingual aligned corpus such as "Canadian Hansard."

(2) Have a native language expert select the most common sense of terms.

(3) Use frequency information from a lexicon that provides its senses with frequency information.

The multilingual concept group n-gram probability database is an optional knowledge database that is constructed from a training data set. The database contents are derived from a text corpus analysis of words used in various supported languages in various contexts. The data in the database can be either (1) sense-correct concept groups assigned to each term in the texts, or (2) all possible concept groups assigned to each term in the texts (e.g., if one term belongs to three concept groups, then three concept groups will be assigned to that term).

This knowledge database collects all concept groups which are assigned to N adjacent terms in the texts. The resulting ordered lists are summed and normalized to produce the likelihood probability of the Nth term assigned with certain concept groups which are assigned to the (N-1)th, . . . (N-(N-1))th terms.

FIG. 3B shows this process where MCGD **160** has had to resort to Domain Knowledge (using the MCGCM) and Global Knowledge (using the n-gram probability database) to disambiguate the polysemous words.

The output of MCGD **160** is a single multilingual concept group for each substantive word in the input text. This concept group may comprise either a single word choice or several word choices, depending on the membership of the concept group. Words from all supported languages will be represented.

5.3 Multilingual Concept Group to Monolingual Hierarchical Concept Mapper (MCG-MHCM) **170**

MCG-MHCM **170** takes as input the fully-tagged, native language text stream with single multilingual concept categories assigned for each substantive word and maps this flat conceptual representation to an English language hierarchical representation. MCG-MHCM **170** performs the following:

(a) Maps all the native language words in a single concept category to the English word member/s in that category.

(b) Converts the English word members of the selected concept group from the multilingual concept database (MCD) to zero or more categories in the monolingual hierarchical concept dictionary (MHCD). This is a static mapping scheme, whereby all the English word members of a particular concept group are treated as being equally likely instantiations. In this static implementation, all English word members of the selected multilingual concept group are mapped to their respective categories in the MHCD. The frequencies of the concept categories mapped to by the English word members of the selected multilingual concept group of a word are summed and the most frequent category for that word is selected. If there are multiple categories in the MHCD to which the English word members of the multilingual concept group map, then these multiple categories need to be disambiguated in the next component of the system.

(c) Maps the many thousand multilingual concept categories to fewer, higher order monolingual categories.

The MHCD is different from the MCD in that the MHCD consists of terms in one language (in the current system, English terms make-up the database). While the MHCD and MCD both define concepts as a groups of synonyms, the MHCD can be characterized by the hierarchical organization which is imposed on the concepts. The hierarchy can be constructed by relating concepts with relations such as "super/sub type" and "broader/narrower." In the current implementation, the MHCD is a COTS product.

The output of the MCG-MHCM module is a tagged, native language text stream with unique, monolingual (English), hierarchical concept categories assigned to each identified substantive word.

5.4 Monolingual Hierarchical Concept Category Disambiguator (MHCD) **180**

MHCD **180** accepts the monolingual categories assigned to substantive words in a text and performs disambiguation similar to that performed by the multilingual concept group disambiguator (MCGD) module. The disambiguation process is similar to the disambiguation performed by the Subject Field Code (SFC) disambiguator covered in "Natural Language Processing."

The MHCD performs the following processing of text using the following evidence sources:

(a) Local Context—The processing here will be nearly identical to the use of local information in MCGD **160** described above. That is, Unique or Frequent categories will be determined for each sentence and then used as "anchors" to select one monolingual category from amongst the multiple monolingual categories to which an ambiguous multilingual concept group has mapped.

(b) Domain Knowledge—The monolingual category correlation matrix (MCCM) is used to indicate the probabilities that the multiple monolingual categories to which a multilingual concept group has been mapped correlate with the Unique or Frequent monolingual category determined by local context. The MCCM is produced from a document corpus, and is similar to the multilingual concept (MCGCM) in terms matrix (MCGCM) in terms of how the two are constructed and their internal structures.

(c) Global Knowledge—If there is no Unique or Frequent monolingual category in an input sentence, then the system has no "anchor" by which to access the Correlation Matrix and must use global knowledge. In this event, the frequency of use of various senses of a word is used as the basis for the global knowledge source.

The output of the MHCD module is a text stream with disambiguated monolingual categories assigned to each substantive word.

5.5 Monolingual Hierarchical Concept Dictionary-Based Vector Generator (MCVG) **190**

MCVG **190** accepts a text stream with single monolingual category assigned to each substantive word in a text, and produces a fixed-dimension vector representation of the concept-level contents of the text. The basic processing performed by this module is the same as that performed by the Subject Field Code (SFC) vector generator described in "Natural Language Processing."

The MCVG generates a representation of the meaning (context) of the text of a document/query in the form of monolingual category (subject) codes assigned to information bearing words in the text. The monolingual category vector for all documents and queries has the same number of dimensions; weights or scores are applied to each dimension according to the presence and frequency of text with certain subject-contents.

The MCVG creates a vector code index file for each document to facilitate efficient searching and matching. Typically, the relative importance of the concept in each document and the link between the term and the document in which the term occurred is preserved. The vector code index file for each document is a fixed length file containing scores/weights for each dimension (called a slot) of the vector.

MCVG **190** performs the following staged processing:

(a) The frequencies of the disambiguated monolingual category codes assigned to words in the text are summed and then normalized in order to control for the effect of document length.

(b) The resulting normalized document vectors are fixed-dimension vectors representing the concept-level contents of the processed text (either documents or queries). They are passed to the next module for either document-to-query-vector matching (comparison), or for document-to-document matching (comparison) for clustering of documents.

5.6 Concept Mapper and Disambiguator Operation

FIGS. **4** and **5** are diagrams showing concrete examples of the processing of French input text to a monolingual concept vector.

FIG. **4** shows the mapping of two substantive French words, "agricole" and "regime." The word "agricole" can be seen to map to a single multilingual concept group with the English language member "agricultural." As can be seen, this multilingual concept group maps to the monolingual category "Agriculture," and contributes to the monolingual category vector, a portion of which is shown schematically at the right side of the figure.

The French word "regime," on the other hand, is polysemous, and maps to three multilingual concept groups (e.g., concept groups with the English language members "reign," "system," and "diet"). The word needs to be disambiguated using the methodology described in the above discussion of MCGD **160**, MCG-MHCM **170** and MHCD modules, such that an unambiguous, single concept code is assigned to the word. In this simple example, since no Local Context or Domain Knowledge can be applied to the disambiguation process by the word "agricole," (and, for the purposes of this example, we assume no other words help in this disambiguation process), Global Knowledge will be applied and the most common sense of the word will be invoked ("system").

FIG. **5** shows a complete single French sentence as input, and shows the two-stage disambiguation explicitly. The native language sentence is shown being processed through the multilingual concept group generation process, to a monolingual conceptual representation with disambiguated concept codes. For simplicity, only the English language members of the multilingual concept groups are shown. In this example, the complete sentence has "anchor codes" (e.g., "comptant," which maps to code **#105**, with the English member "in cash") that can be used to help disambiguate other polysemous words in the sentence using Local or Domain processing. For example, the French "les paiements" maps to three codes, which are disambiguated at the MCGD to a Finance code).

By way of background, FIG. **6** shows an example of a portion of the processing in a monolingual system such as described in "Natural Language Processing." In particular, FIG. **6** shows the SFC system for monolingual vector representation of the conceptual contents of a document.

6.0 Generation of Term-Based Representations

6.1 Probabilistic Term Indexer (PTI) **210**

PTI **210** accepts the output from PNC **140** (documents only) and creates a new appended field in the document index file. The PTI also assigns a weighted, TF.IDF score (the product of Term Frequency and Inverse Document Frequency) for each proper noun. This could be applied to other types of terms. This weighted score is used in QDM and score combiner **230**. This index file contains all proper nouns and their associated TF.IDF scores.

PTI **210** assigns TF.IDF scores for each proper noun as follows:

$$TF*IDF=(ln(TF)+1)*ln(N+1/n)$$

where TF is the number of occurrences of a term within a given document, IDF is the inverse of the number of documents in which the term occurs, compared to the whole corpus, N is the total number of documents in the corpus, and n is the number of documents in which the term occurs. The product of TF.IDF provides a quantitative indication of a term's relative uniqueness and importance for matching purposes. TF.IDF scores are calculated for documents and queries. The IDF scores are based upon the frequency of occurrence of terms within a large, representative sample of documents in each supported language.

The output of the PTI is an index of proper nouns and expansions with associated TF.IDF scores.

6.2 Probabilistic Query Processor (PQP) **220**

PQP **220** accepts the native-language query with disambiguated concept group assignments for each substantive word in the query from MCGD **160** and performs the following processing:

(a) Negation

It is common for queries to simultaneously express both items of interest and those items that are not of interest. For example, a query might be phrased "I am interested in A and B, but not in C." In this instance, A and B are required (they are in the "positive" portion of the query) and C is negated and not required (it is in the negative portion of the query). Only terms in the positive portion of the query are considered for document matching. The PQP uses the principles of text structure analysis and models of discourse to identify the disjunction between positive and negative portions of a query. The principles employed to identify the positive/negative disjunction are based on the general observation among discourse linguists that writers are influenced by the established schema of the text-type they produce, and not just on the specific content they wish to convey. This established schema can be delineated and used to computationally instantiate discourse-level structures. In the case of the discourse genre of queries written for online retrieval systems, empirical evidence has established several techniques for locating the positive/negative disjunction.

(a1) Lexical Clues

For each supported language there exists a class of frequently used words or phrases that, when connected in a logical sequence, are used to establish the transition from the positive to the negative portion of the query (or the reverse). In English such a sequence might be as simple as "I am interested in" followed by ", but not." Clue words or phrases must have a high frequency of occurrence within the confines of a particular context.

(a2) Component Ordering

Components in a query tend to occur in a certain repetitive sequence, and this sequence can be used as a clue to establish negation.

(a3) Continuation Clues

Especially in relatively long queries a useful clue for negation disjunction detection across sentence boundaries is conjunctive relations which occur near the beginning of a sentence and which have been observed in tests to predictably indicate possible transitions from sentence to sentence.

(b) Construction of Logical Representation of the Query

A tree structure with terms connected by logical operators is constructed using a native-language sublanguage processor.

FIG. 7 shows the tree representation of the following query:

"I am interested in any information about A and B and C, D or E and F."

The latter portion of the query can be represented as:

A and B and (C or D or (E and F)).

The tree structure includes a head term, which can be a Boolean AND or OR operator (AND in this case), which links, possibly through intermediate nodes, to extracted query terms at terminal nodes (A, B, C, D, E, and F). The intermediate nodes are also Boolean AND or OR operators.

Various lexical clues are used to determine the logical form of the query. The basis of this system is a sublanguage grammar which is based on probabilistic generalizations regarding the regularities exhibited in a large corpus of query statements. The sublanguage relies on items such as function words (the placement of articles, auxiliaries and prepositions), meta-text phrases, and punctuation (or the combination of these elements) to recognize and extract the formal logical combination of relevancy requirements from the query. The sublanguage interprets the query into pattern-action rules which reveal the combination of relations that organize a discourse, and which allow the creation from each sentence of a first-order logic assertion, reflecting the Boolean assertions in the text.

Part of this sublanguage is a limited anaphor resolution (that is, the recognition of a grammatical substitute, such as a pronoun or pro-verb, that refers back to a preceding word or group of words). An example of a simple anaphoric reference is shown below:

"I am interested in the stock market performance of IBM. I am also interested in the company's largest foreign shareholders."

In this example, the phrase "the company's" is an anaphoric reference back to "IBM."

A summary of the fuzzy Boolean operators and their function is shown in Table 2, below.

TABLE 2

Logical Operators Used in Sublanguage Processing

Operator	Operation	Fuzzy weight/Score
AND	Boolean AND	Addition of scores within AND operator
OR	Boolean OR	Maximum score from all ORed terms
!NOT	Negation	—

Each term in the logical representation is assigned a weighted score. Scores are normalized such that the maximum attainable score during matching (if all terms are successfully matched with a document) is 1.0. During matching the fuzzy logical AND operator performs an addition with all matched ANDed term scores. The fuzzy OR operator selects the highest weighted score from among all the matched ORed terms. For example, in the query representation of FIG. 7, if terms A, C and F are matched, then the score assigned the match would be 0.66 (that is, 0.33 from the match with A, and 0.33 from the match with C, which is the higher of the ORed C and F weighted scores).

The negation operator (!NOT) divides the query into two logical portions: the positive portion of the query contains all positive assertions in the query statement; the negative portion of the query contains all the negative assertions in the query. No score is assigned to this operation.

The output of the PQP is a logical representation of the query requirements with fuzzy Boolean weights assigned to all terms.

7.0 Matching Documents with Queries

Documents and queries are processed for matching in their English language form to take advantage of the monolingual processing modules of the DR-LINK information retrieval system [Liddy 94a]; [Liddy 94b]; [Liddy 95].

Documents are arranged in ranked order according to their relative relevance to the substance of a query. The matcher uses a variety of evidence sources to determine the similarity or suitable association between query and documents. Various representations of document and query are used for matching, and each document-query pair is assigned a match score based on (1) the distance between vectors, and (2) the frequency and occurrence of proper nouns.

The fact that the documents are represented in a common, language-independent vector format of weighted slot values, no matter what the language of the individual documents, enables the system to treat all documents similarly. Therefore, it can: (1) cluster documents based on similarity amongst them, and (2) provide a single list of documents ranked by relevancy, with documents of various languages interfiled. Thus the process whereby documents are retrieved and ranked for review by the user is language independent.

7.1 Monolingual Category Vector Matcher (MCVM) 200

MCVM 200 is similar to the Subject Field Code (SFC) matcher described in "Natural Language Processing."

The process of document to query matching using the monolingual category vector is:

(a) Generation of the monolingual category vector for query and document (see earlier discussion and FIGS. 3A and 3B).

(b) Generation of distance/proximity measures. The vector for each text is normalized in order to control for the effect of document length. The vector codes can be considered a special form of controlled vocabulary (all words and terms are reduced to a finite number of vector codes). A similarity measure of the association or correlation of the

query and document vectors is assigned by simulating the distance/proximity of the respective vectors in multi-dimensional space using similarity measure algorithms.

7.2 Query to Document Matcher (QDM) and Score Combiner **230** QDM and score combiner **230** accepts three input streams: the TF.IDF scores for documents from the document index created by PTI **210**; the logical query representation from PQP **220**; and the vector representation of both document and query from the MCVM **200**. The output of the QDM and score combiner module is a score representing the match between documents and query.

Using the evidence sources listed above, the matcher determines the similarity or suitable association between the query and the documents. Various representations of document and query are used for matching. Each document-query pair is assigned a series of match scores based on (1) the common occurrence of proper nouns or expansions in the logical query representation, (2) TF.IDF scores, and (3) the distance between vectors.

Documents are assigned scores using the following evidence:

(a) Monolingual Category Vectors. The proximity of the vector for query and document.

(b) Positive TF.IDF (TF.IDF for the positive portion of the query). Matching is based on a natural-log form of the equation TF.IDF, where TF is the number of occurrences of a term within a given document, and IDF is the inverse of the number of documents in which the term occurs, compared to the whole corpus (see description of PTI **210**). The scores are normalized to the highest TF.IDF score for all documents.

(c) Query match. The matching of proper nouns (or other terms) and expansions scored from the logical query representation.

7.3 Document Scores

A logistic regression analysis using a Goodness of Fit model is applied to compute a relevance score for each document. Three independent variables, corresponding to the three types of evidence mentioned above, are used.

Regression coefficients for each variable in the regression equation are calculated using an extensive, representative, multilingual test corpus of documents for which relevance assignments to a range of queries have been established by human judges.

The logistic probability (logprob) of a given event is calculated as follows:

$$\text{logprob (event)} = 1/(1+e^{-Z})$$

where Z is the linear combination

$$Z = B_o + B_1 B_1 + B_2 X_2 + B_3 X_3$$

and $B_{1\text{-}3}$ are the regression coefficients for the independent variables $X_{1\text{-}3}$. Documents are ranked by their logistic probability values, and output with their scores.

8.0 Presentation of Results

8.1 Recall Predictor **240**

The matching of documents to a query organizes documents by matching scores in a ranked list. The total number of presented documents can be selected by the user or the system can determine a number using the Recall Predictor (RP) function. Note that documents from different sources are interfiled and ranked in a single list.

The RP filtering function is accomplished by means of a multiple regression formula that successfully predicts cut-off criteria for individual queries based on the similarity of documents to queries as indicated by the vector matching (and preferably the proper noun matching) scores. The RP is sensitive to the varied distributions of similarity scores (or match scores) for different queries, and is able to present to the user a certain limited percentage of the upper range of scored documents with a high probability that close to 100% recall will be achieved. The user is asked for the desired level of recall (up to 100%), and a confidence interval on the retrieval. While in some cases a relatively large portion of the retrieved documents would have to be displayed, in most cases for 100% recall with a 95% confidence interval less than 20% of the retrieved document collection need be displayed. In trials of the DR-LINK system (level of recall 100%, confidence level 95%), the system has collected an average of 97% of all documents judged relevant for a given query [Liddy 94b].

8.2 Graphical User Interface (GUI) **250**

GUI **250** uses clustering techniques to display conceptually-similar documents. The GUI also allows users to interact with the system by invoking relevance feedback, whereby a selection of documents or a single document can be used as the basis for a reformulated query to find those documents with conceptually similar contents.

The GUI for the CINDOR system is specifically intended to be suitable for users of any nationality, even if their knowledge of foreign languages is sparse. Graphic representations of documents will be used, with textual/descriptive representations kept to a minimum. Research has shown that the factors that influence comprehension of new data are (1) the rate at which information is presented, (2) the complexity of the information, and (3) how meaningful the new information is. Highly meaningful information is accepted with relative ease; less meaningful information, in addition to being less useful, requires greater cognitive effort to comprehend (and usually reject). Coherence of presentation and an association with existing knowledge are both highly correlated with increased meaningfulness. Thus the concept behind the user interface is to present "details on demand," showing only enough information to allow quick apprehension of relevance: more details are immediately available though hypertext links.

8.3 Document Clustering, Browsing and Relevance Feedback

The monolingual category vectors are used as the basis for the clustering and display, and for the implementation of relevance feedback in the system:

8.3.1 Clustering

Documents can be clustered using an agglomerative (hierarchical) algorithm that compares all document vectors and creates clusters of documents with similarly weighted vectors. The nearest neighbor/Ward's approach is used to determine clusters, thus not forcing uniform sized clusters, and allowing new clusters to emerge when documents reflecting new subject areas are added. These agglomerative techniques, or divisive techniques, are appropriate because they do not require the imposition of a fixed number of clusters.

Using the clustering algorithm described above, or other algorithms such as single link or nearest neighbor, CINDOR is capable of mining large data sets and extracting highly relevant documents arranged as conceptually-related clusters in which documents from several languages co-occur.

Headlines from newspaper articles or titles from documents in the cluster are used to form labels for clusters. Headlines or titles are selected from documents that are near the centroid of a particular cluster, and are therefore highly representative of the cluster's document contents. An alternative labeling scheme, selectable by the user, is the use of

the labeled subject codes which make up either the centroid document's vector or the cluster vector.

The user is able to browse the documents, freely moving from cluster to cluster with the ability to view the full documents in addition to their summary representation. The user is able to indicate those documents deemed most relevant by highlighting document titles or summaries. If the user so decides, the relevance feedback steps can be implemented and an "informed" query can be produced, as discussed below.

The CINDOR system is thus able to display a series of conceptually-related clusters in response to a browsing query. Each cluster, or a series of clusters, could be used as a point of departure for further browsing. Documents indicative of a cluster's thematic and conceptual content would be used to generate future queries, thereby incorporating relevance feedback into the browsing process. The facility for browsing smaller, semantically similar sub-collections which contain documents of multiple languages aids users in determining which documents they might choose to have translated.

8.3.2 Developing "Informed" Queries for Relevance Feedback

Relevance feedback is accomplished by combining the vectors of user-selected documents or document clusters with the original query vector to produce a new, "informed" query vector. The "informed" query vector will be matched against all document vectors in the corpus or those that have already passed the cut-off filter. Relevant documents will be re-ranked and re-clustered.

1. Combining of Vectors. The vector for the original query and all user-selected documents are weighted and combined to form a new, single vector for re-ranking and re-Clustering.

2. Re-Matching and Ranking of Corpus Documents with New, "Informed" Query Vector. Using the same similarity measures described above for MCVM **200**, the "informed" query vector is compared to the set of vectors of all documents above the cut-off criterion produced by the initial query (or for the whole corpus, as desired), then a revised query-to-document concept similarity score is produced for each document. These similarity scores are the system's revised estimation of a document's predicted relevance. The set of documents are thus re-ranked in order of decreasing similarity of each document's revised predicted relevance to the "informed" query on the basis of revised similarity value.

3. Cut-Off and Clustering after Relevance Feedback. Using the same regression formula described above in connection with recall predictor **240**, a revised similarity score cut-off criterion is determined by the system on the basis of the "informed" query. The regression criteria are the same as for the original query, except that only the vector similarity score is considered. The agglomerative (hierarchical) clustering algorithm is applied to the vectors of the documents above the revised cut-off criterion and a re-clustering of the documents will be performed. Given the re-application of the cut-off criterion, the number of document vectors being clustered will be reduced, and improved clustering is achieved.

8.4 Application of "Gloss" Transliteration to Highly Relevant Documents

Conceptual-level matching and disambiguation of words ensures that when these words are translated, the correct sense or meaning will be selected. It is therefore possible to offer a surface-level transliteration of highly relevant documents with a very high degree of certainty that the correct translation of words will be performed.

An example of the transliteration system output is shown below:

French Original Text:	Les Surplus et les chutes des prix agricole entrainent
CINDOR Transliteration:	rise fall price agricultural bring about
English Translation:	The rise and fall of agricultural prices drives
French Original Text:	des mouvements sur les marches. La faute a qui? . . .
CINDOR Transliteration:	movements markets. fault who? . . .
English Translation:	movements in the markets. Whose fault is it? . . .

Only some of the words will be mapped into corresponding, disambiguated words or phrases in another language. Much of the text in a document, especially the functional classes of words, will remain un-transliterated. Indeed, one of the strengths of this approach is that the laborious and expensive process of translating a great many foreign documents to ascertain relevance can be avoided. With CINDOR, only those few documents that obtain a high relevance ranking and show promise in their transliterated form become candidates for full translation, if desired. The selection of words could be based on (1) whether they have been indexed in the MCD, (2) their POS-tag assignment, (3) anaphoric disambiguation, and (4) meta-textual and discourse-level considerations, such as whether words and phrases are in the headline of a text.

8.5 Machine Translation of Relevant Documents

Documents or document clusters that, based on their high relevance ranking, the gloss transliteration, or other factors, are deemed to be highly relevant to a query, and are candidates for a machine translation of the original foreign language text. CINDOR thus ensures that only those few documents that are especially pertinent to a query will undergo the full translation process.

CINDOR incorporates a range of computer aided translation modules, each a COTS technology, that translate a given document from one language to another. The selection of the appropriate COTS module is automatic, being based on the language identification assignment for each document provided by LI **120** and on the identified language of the query. For any given query and range of documents it is likely that multiple translation modules will be activated.

Each machine translation COTS module, or MT engine, will process source documents to create a given translation without human intervention or aid. In cases where the document contains arcane or industry-specific terminology, such as with medical or legal documents, multilingual mapping terminology managers with objects stored in a conceptual orientation may also be invoked to aid the translation process.

9.0 References

[Liddy94a] Liddy, E. D. & Myaeng, S. H. (1994). DR-LINK System: Phase I Summary. *Proceedings of the TIPSTER Phase I Final Report.*

[Liddy94b] Liddy, E. D., Paik, W., Yu, E. S. & McKenna, M. (1994). Document retrieval using linguistic knowledge. *Proceedings of RIAO '94 Conference.*

[Liddy95] Liddy, E. D., Paik, W., McKenna, M. & Yu, E. S. (1995). A natural language text retrieval system with relevance feedback. *Proceedings of the 16th National Online Meeting.*

10.0 Conclusion

In conclusion, it can be seen that the present invention provides an elegant and efficient tool for multilingual docu-

ment retrieval. The system permits even those searchers with limited or no knowledge of foreign languages to gather highly relevant information from international sources. Since the system offers a "gloss" transliteration of target texts, the user is able to ascertain relevance of foreign-language texts so as to be able to make an intelligent decision regarding full translation.

While the above is a complete description of specific embodiments of the invention, various modifications, alternative constructions, and equivalents may be used. For example, while the specific embodiment augments concept level matching through the use of term-based representations and matching, it is possible to implement an embodiment using concept level matching alone. Additionally, evidence combination criteria could be modified for different retrieval criteria. For example, some specific terms or some specific concept categories may be considered mandatory for matching, such that matching would be a two-step process of foldering based on logical requirements, and within folders regression-based matching scores would be used.

Similarly, while the described disambiguation method is the presently preferred method, there are other possibilities, such as statistical or entirely probabilistic techniques. Indeed, disambiguation of concept codes, while preferred, is not essential. Moreover, the concept vector categories, codes, and hierarchy could be modified or expanded, as could the proper noun categories, codes, and hierarchy.

Another language-independent method of representing text is using n-gram coding, wherein a text is decomposed to a sequence of character strings, where each string contains n adjacent characters from the text. This can be done by moving an n-character window n characters at a time, or by moving the n-character window one character at a time. In an n-gram representation, no attempt is made to understand, interpret or otherwise catalog the meaning of the text, or the words that make up the text. A tri-gram representation is the special case where n=3. Representation and matching are based on the co-occurrence of n-grams or a sequence of character strings, or on the co-occurrence and relative prevalence of such n-grams, or on other, similar schemes. Such analysis is an alternative representational scheme for CINDOR.

In this alternative embodiment, an n-gram query processor (NQP) module replaces probabilistic query processor (PQP) **220**, an n-gram document processor replaces probabilistic term Indexer (PTI) **210**, and an n-gram query to document matcher replaces query_to_document_matcher (QDM **230**). The NQP accepts the native-language input and performs the following processing: a) decomposes each term in the queries into n-adjacent-character strings; and b) lists each unique n-adjacent-character string with the number of occurrences as the document representation. The NDP accepts the output from PNC **140** and performs the following processing: a) decomposes each term in the document into n-adjacent-character strings; and b) lists each unique n-adjacent-character string with the number of occurrences as the query representation. The NQDM accepts two input streams, namely the outputs from the NQP and NDP, and provides a score representing the match between the documents and query. This output is an input to the score combiner. Documents are assigned scores by measuring the degrees of overlap between the n-gram decomposed terms from documents and queries. The larger the overlap, the higher the degree of relevance.

Therefore, the above description should not be taken as limiting the scope of the invention as defined by the claims.

What is claimed is:

1. A method of representing documents in a database that includes documents in a plurality of languages, the method carried out for each document, comprising:

determining a set of potential conceptual-level meanings of at least some words in the document from a language-independent multilingual concept database that reflects the plurality of languages and comprises a collection of concept groups;

mapping the sets of potential conceptual-level meanings, so determined, to respective single language-independent conceptual-level meanings; and

generating a language-independent conceptual representation of the subject content of the document based on the language-independent conceptual-level meanings determined in said mapping step.

2. The method of claim **1**, and further comprising, determining the language of the document for at least some documents.

3. The method of claim **1**, and further comprising

generating a term-based representation of the document for at least some documents.

4. The method of claim **3** wherein the term-based representation of the document is a representation of a set of proper nouns found in the document.

5. The method of claim **4** wherein the set of proper nouns found in the document are represented as categories from a hierarchical classification scheme.

6. The method of claim **3** wherein the term-based representation of the document is a representation of a set of noun phrases found in the document.

7. The method of claim **1** wherein a given one of said words is polysemous, giving rise to multiple conceptual-level meanings from the multilingual concept database, and said mapping, for the given word, comprises:

disambiguating among the multiple conceptual-level meanings from the multilingual concept database to provide a single multilingual conceptual-level meaning;

mapping the single multilingual conceptual-level meaning to a set of monolingual concept categories in a monolingual concept dictionary; and

if the set of monolingual concept categories from the monolingual concept dictionary contains multiple monolingual concept categories, disambiguating among the multiple monolingual concept categories to provide the single language-independent conceptual-level meaning.

8. The method of claim **7** wherein disambiguating includes:

analyzing local context information to attempt to determine a single meaning;

if a single meaning is not determined from analyzing local context information, analyzing domain knowledge to attempt to determine a single meaning; and

if a single meaning is not determined from analyzing domain knowledge, analyzing global information to attempt to determine a single meaning.

9. The method of claim **7**, and further comprising

providing a gloss transliteration using the single multilingual conceptual-level meaning derived from disambiguating among the multiple conceptual-level meanings.

10. The method of claim **1** wherein the collection of concept groups includes words or phrases, from the plurality of languages, that are conceptually synonymous.

11. The method of claim 1, and further comprising:

determining a measure of proximity of the language-independent conceptual representation of each document to the language-independent conceptual representation of the other documents in the plurality; and

clustering the documents in the plurality according to the documents' respective measures of proximity to each other.

12. A method of retrieving documents in response to a query, the query being in a user-selected language of a plurality of languages, the method comprising:

providing a corpus of documents, each in a language of said plurality of languages, at least one of the documents being in a language other than the user-selected language;

for each document;

determining a set of multilingual concepts of at least some words in the document using a language-independent multilingual concept database that reflects the plurality of languages and comprises a collection of concept groups;

mapping the sets of multilingual concepts, so determined, to respective single language-independent conceptual-level meanings; and

generating a language-independent conceptual representation of the subject content of the document based on the language-independent conceptual-level meanings determined in said mapping;

generating a language-independent conceptual representation of the subject content of the query; and

for each document, generating a measure of relevance of the document to the query using the conceptual representation of the subject content of the document and the conceptual representation of the subject content of the query.

13. The method of claim 12 wherein the query is a natural language query.

14. The method of claim 12 wherein generating a language-independent conceptual representation of the subject content of the document comprises:

generating a conceptual-level vector representing the subject content of the document.

15. The method of claim 12 wherein;

the concept groups include a collection of synonyms and near-synonyms of a given word or phrase in said plurality of languages;

a given word in the document is polysemous, giving rise to multiple conceptual-level meanings from the multilingual concept database; and

for the given word, mapping the sets of multilingual concepts to respective single language-independent conceptual-level meanings comprises:

disambiguating the multiple conceptual-level meanings.

16. The method of claim 15 wherein disambiguating the multiple conceptual-level meanings comprises:

disambiguating among the multiple conceptual-level meanings from the multilingual concept database to provide a single multilingual conceptual-level meaning;

mapping the single multilingual conceptual-level meaning to a set of monolingual concept categories in a monolingual concept dictionary; and

if the set of monolingual concept categories from the monolingual concept dictionary contains multiple monolingual concept categories, disambiguating among the multiple monolingual concept categories to provide the single language-independent conceptual-level meaning.

17. The method of claim 16 wherein disambiguating includes:

analyzing local context information to attempt to determine a single meaning;

if a single meaning is not determined from analyzing local context information, analyzing domain knowledge to attempt to determine a single meaning; and

if a single meaning is not determined from analyzing domain knowledge, analyzing global information to attempt to determine a single meaning.

18. The method of claim 16, and further comprising

providing a gloss transliteration using the single multilingual conceptual-level meaning derived in disambiguating among the multiple conceptual-level meanings.

19. The method of claim 12, and further comprising providing a gloss transliteration of at least some of the words in at least one of the documents.

20. The method of claim 12 wherein said language-independent conceptual representation of the subject content of the document is augmented by a language-dependent statistical index using words in the document's language as indexing units.

21. The method of claim 12 wherein said language-independent conceptual representation of the subject content of the document includes a statistical index using N-gram style decomposed words as indexing units.

22. The method of claim 12 wherein generating a language-independent conceptual representation of the subject content of the query comprises:

mapping words or phrases in the query into language-independent concepts; and

generating a conceptual-level vector representing the subject content of the query.

23. The method of claim 12 wherein said language-independent conceptual representation of the subject content of the query includes N-gram style decomposed terms as language-independent query requirements.

24. The method of claim 12 wherein said language-independent conceptual representation of the subject content of the query is augmented using a language-dependent logic requirement, the logic requirement including terms and logical connectives where the terms include the query term and its synonymous terms in the multilingual concept database.

25. The method of claim 12, and further comprising:

providing a list of at least some of the documents;

receiving user input specifying at least one document, or a part thereof, on the list; and

generating a revised query representation based on the original query plus a representation of the specified document or documents, or parts thereof.

26. The method of claim 12, and further providing a relevance-ranked list of at least some of the documents.

27. The method of claim 26, wherein the number of documents in the relevance-ranked list of documents is calculated based on a user-specified level of recall.

28. The method of claim 26, wherein the number of documents in the relevance-ranked list of documents is calculated based on a user-specified level of recall and a user-specified level of confidence in that level of recall.

29. The method of claim 26, wherein retrieved documents in the relevance-ranked list of documents are ranked without regard to the language they are written in.

30. The method of claim 12, and further determining the language of the document before generating a language-independent representation of the document.

31. The method of claim 12 wherein generating a measure of relevance for a given document comprises:

generating conceptual-level vectors for the given document and for the query; and

determining a distance between the vectors, the distance representing the measure of relevance, with a smaller distance representing a higher degree of relevance.

32. The method of claim 12 wherein generating a measure of relevance for a given document comprises:

generating an N-gram decomposed term representation for the given document and for the query; and

determining a degree of overlap between the N-gram decomposed terms, the overlap representing the measure of relevance, with a larger overlap representing a higher degree of relevance.

33. The method of claim 12 wherein generating a measure of relevance for a given document comprises:

generating word representations for the given document and for the query;

organizing words in the query as logical requirements; and

determining a coverage of terms in the documents against the logical requirement of a query, the coverage representing the measure of relevance, with a larger coverage representing a higher degree of relevance.

34. A method of retrieving documents in response to a query in a user-selected language of a plurality of languages, the method comprising:

(a) providing a corpus of documents, each in a language of said plurality of languages, at least some of the documents being in a language other than the user-selected language;

(b) processing each document by

determining the language of the document,

determining conceptual-level meaning of at least some words in the document from a language-independent multilingual concept database comprising a collection of concept groups,

mapping the conceptual-level meanings into language-independent concepts, and

generating a conceptual-level vector representing the subject content of the document;

(c) processing the query by

mapping words or phrases in the query into language-independent concepts, and

generating a conceptual-level vector representing the subject content of the query; and

(d) for each document, determining a measure of relevance to the query.

35. The method of claim 34 wherein mapping the conceptual-level meanings into language-independent concepts comprises

disambiguating multiple senses of polysemous words and phrases to generate the language-independent concepts.

* * * * *

US006721729B2

(12) **United States Patent**
Nguyen et al.

(10) **Patent No.: US 6,721,729 B2**
(45) **Date of Patent: Apr. 13, 2004**

(54) **METHOD AND APPARATUS FOR ELECTRONIC FILE SEARCH AND COLLECTION**

(76) Inventors: **Thanh Ngoc Nguyen**, 558 N. Wrightwood Dr., Orange, CA (US) 92869; **William Norman Morris, Jr.**, 21512 Camino Papal, Lake Forest, CA (US) 92630; **Phu Thien Ngo**, 771 S. Lassen Ct., Anaheim, CA (US) 92804

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 166 days.

(21) Appl. No.: **09/876,901**

(22) Filed: **Jun. 11, 2001**

(65) **Prior Publication Data**

US 2002/0023077 A1 Feb. 21, 2002

Related U.S. Application Data

(60) Provisional application No. 60/210,482, filed on Jun. 9, 2000.

(51) **Int. Cl.**[7] **G06F 17/30**
(52) **U.S. Cl.** **707/3**; 707/4; 707/5
(58) **Field of Search** 707/1, 2, 3, 4, 707/5, 6, 10, 100

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,321,833 A		6/1994	Chang et al.	395/600
5,594,837 A	*	1/1997	Noyes	706/55
5,768,578 A	*	6/1998	Kirk et al.	707/100
5,794,178 A	*	8/1998	Caid et al.	704/9
5,794,237 A		8/1998	Gore, Jr.	707/5
5,806,060 A	*	9/1998	Borgida et al.	707/3
5,809,484 A		9/1998	Mottola et al.	705/38
5,924,090 A		7/1999	Krellenstein	707/5
5,974,405 A	*	10/1999	McGuinness et al.	706/45
6,012,053 A		1/2000	Pant et al.	707/3
6,052,122 A		4/2000	Sutcliffe et al.	345/331
6,064,985 A		5/2000	Anderson	705/36

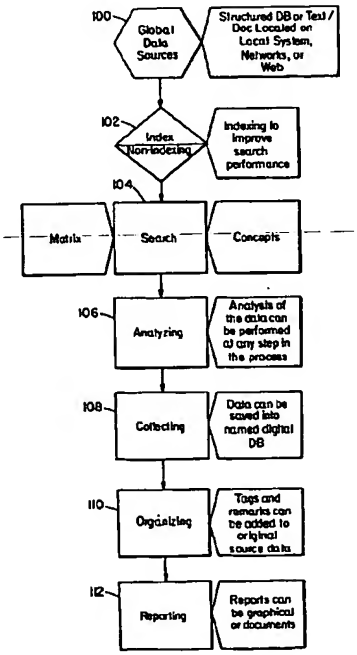
* cited by examiner

Primary Examiner—Diane D. Mizrahi
(74) *Attorney, Agent, or Firm*—Patton Boggs LLP

(57) **ABSTRACT**

A integrated method for searching and reporting the search of electronic data files by receiving a plurality of first and second search concepts from the user, forming the first and second concepts into two-dimensional matrix of paired concepts, performing a search of one or more databases based on all concepts and paired concepts in the matrix, and identifying and displaying a corresponding matrix of search results. An integrated search collection provides formatted documents for drag and drop collection of search information an construction of a search library. An integrated report generation utilizes the format of the collection document for automatic construction of a report.

9 Claims, 12 Drawing Sheets



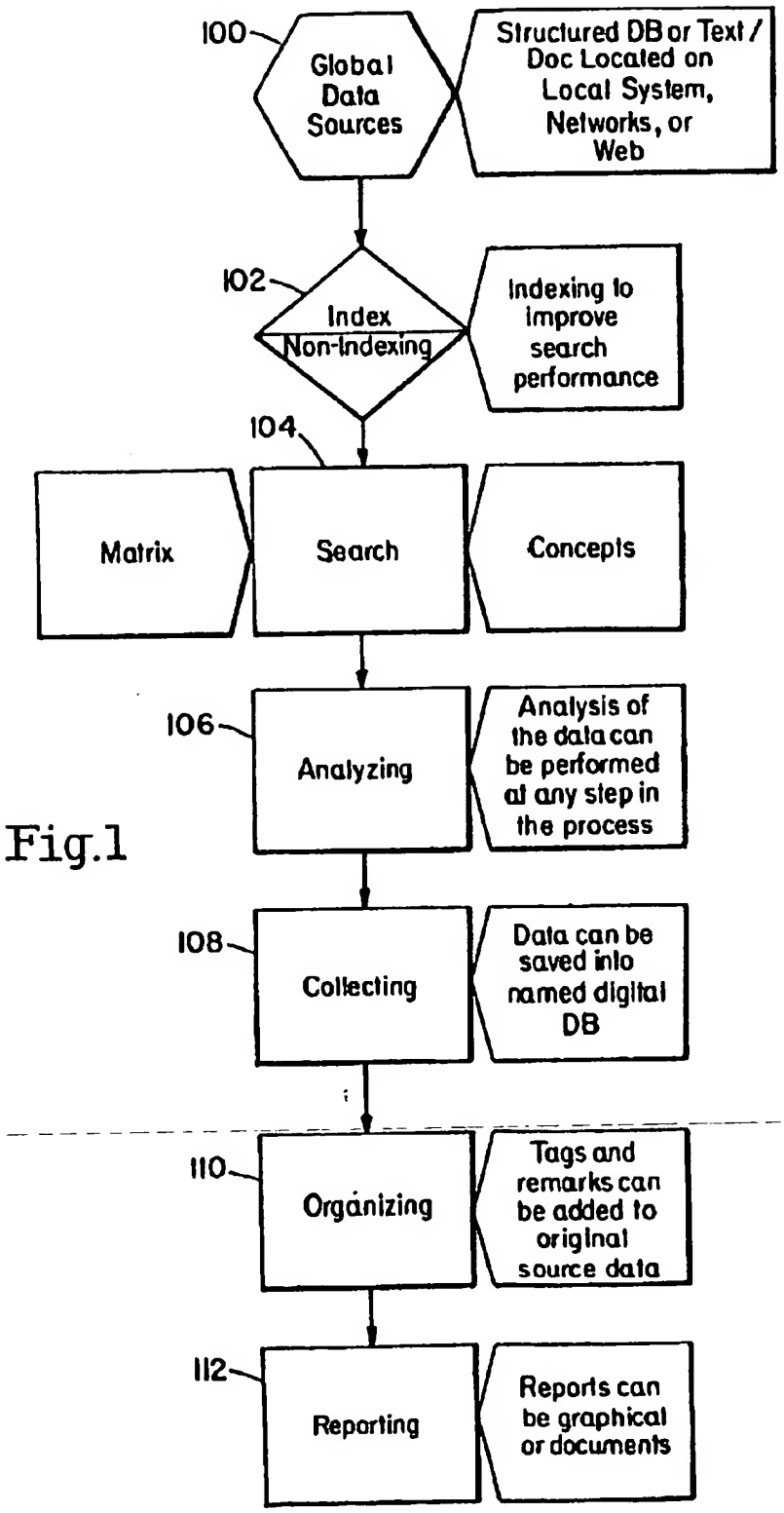

100
Global Data Sources
Structured DB or Text / Doc Located on Local System, Networks, or Web
102
Index
Non-Indexing
Indexing to improve search performance
104
Matrix
Search
Concepts
106
Analyzing
Analysis of the data can be performed at any step in the process
108
Collecting
Data can be saved into named digital DB
110
Organizing
Tags and remarks can be added to original source data
112
Reporting
Reports can be graphical or documents

Fig.1

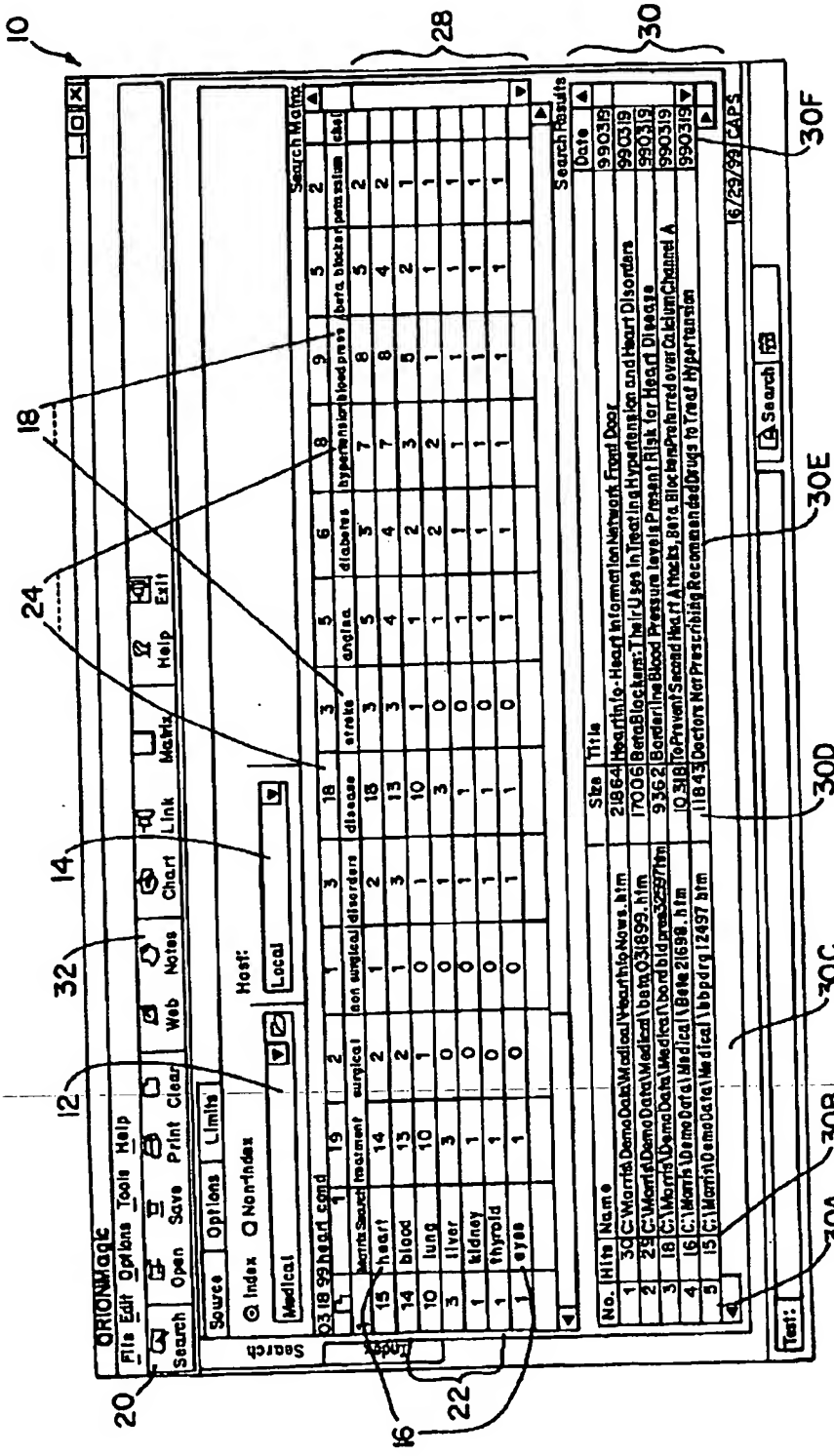

10
28
30
18
24
14
32
12
20
16
22
30A
30B
30C
30D
30E
30F
ORIONImage
File Edit Options Tools Help
Search
Open
Save
Print
Clear
Limits
Source
Options
Web
Notes
Chart
Link
Matrix
Help
Exit
Host:
Local
Medical
Index
Nonindex
Search
Search Matrix
Search Results
Test:
Search
6/29/99 CAPS

Fig. 2

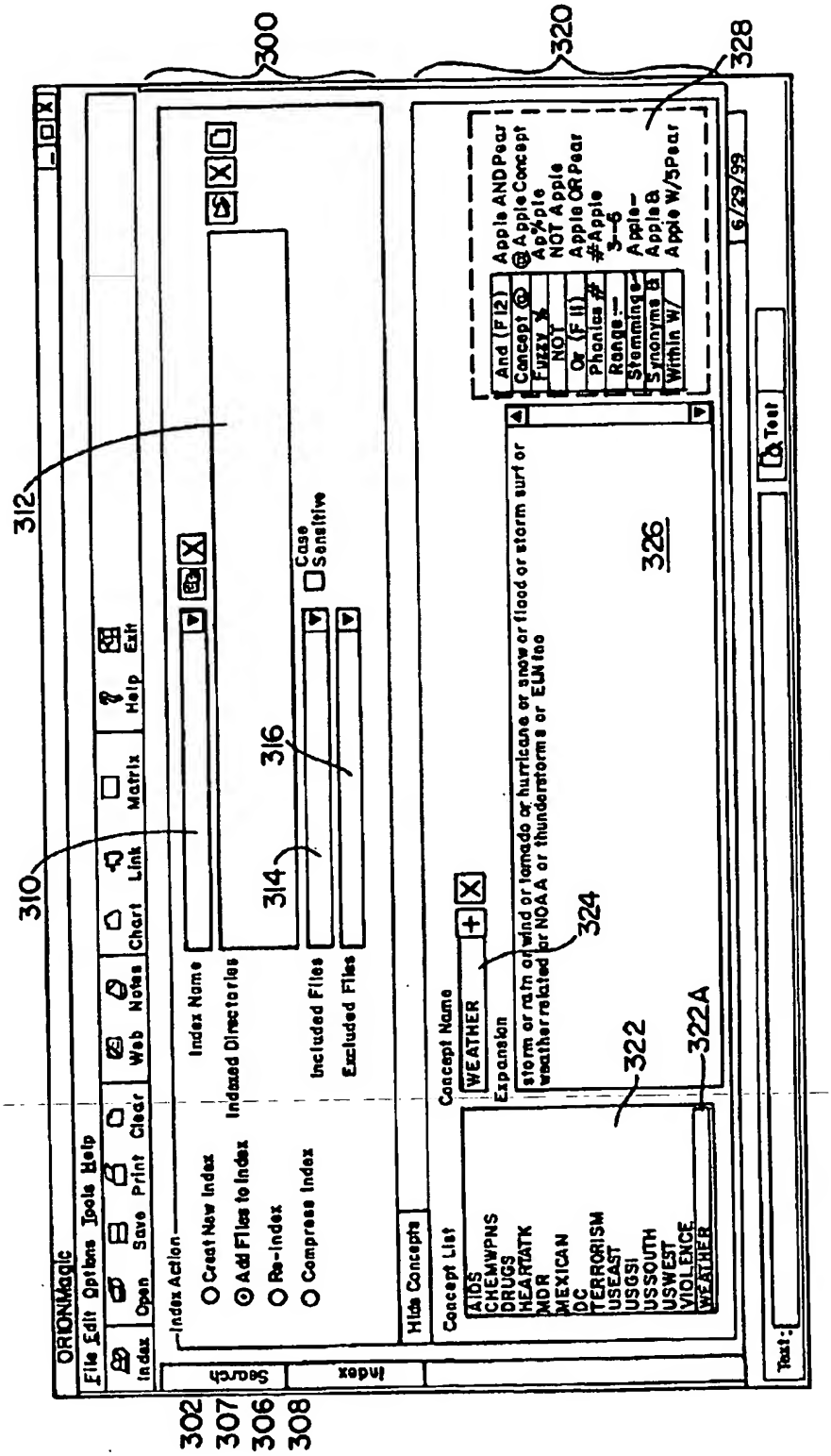

300
320
328
312
310
316
314
326
324
322
322A
302
307
306
308
ORIONMagic
File Edit Options Tools Help
Index
Search
Index
Open
Save
Print
Clear
Web Notes
Chart
Link
Matrix
Help
Exit
Index Action
Create New Index
Add Files to Index
Re-Index
Compress Index
Index Name
Indexed Directories
Included Files
Excluded Files
Case Sensitive
Hide Concepts
Concept List
AIDS
CHEMWPNS
DRUGS
HEARTATK
MDR
MEXICAN
OC
TERRORISM
USEAST
USGSI
USSOUTH
USWEST
VIOLENCE
WEATHER
Concept Name
WEATHER
Expansion
storm or rain or wind or tornado or hurricane or snow or flood or storm surf or weather related or NOAA or thunderstorm or ELNino
And (F12)
Concept @
Fuzzy %
NOT
Or (F11)
Phonics #
Range--
Stemming $
Synonyms &
Within W/
Apple AND Pear
@Apple Concept
Ap%ple
NOT Apple
Apple OR Pear
#Apple
3--6
Apple--
Apple&
Apple W/3Pear
6/29/99
Tool
Text:

Fig.3

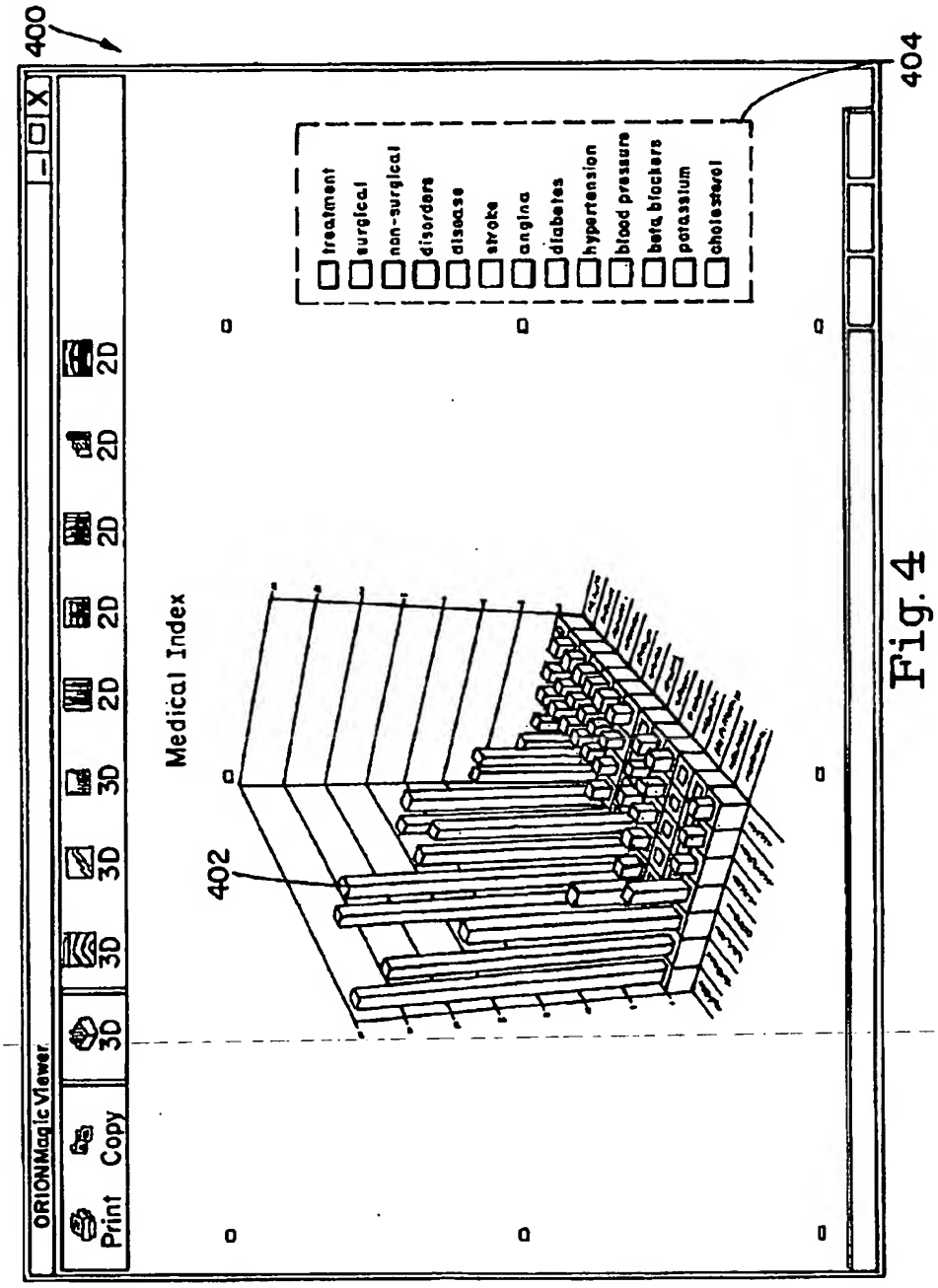

400
404
402
treatment
surgical
non-surgical
disorders
disease
stroke
angina
diabetes
hypertension
blood pressure
beta blockers
potassium
cholesterol
Medical Index
ORION Magic Viewer
Print
Copy
3D
3D
3D
3D
2D
2D
2D
2D

Fig. 4

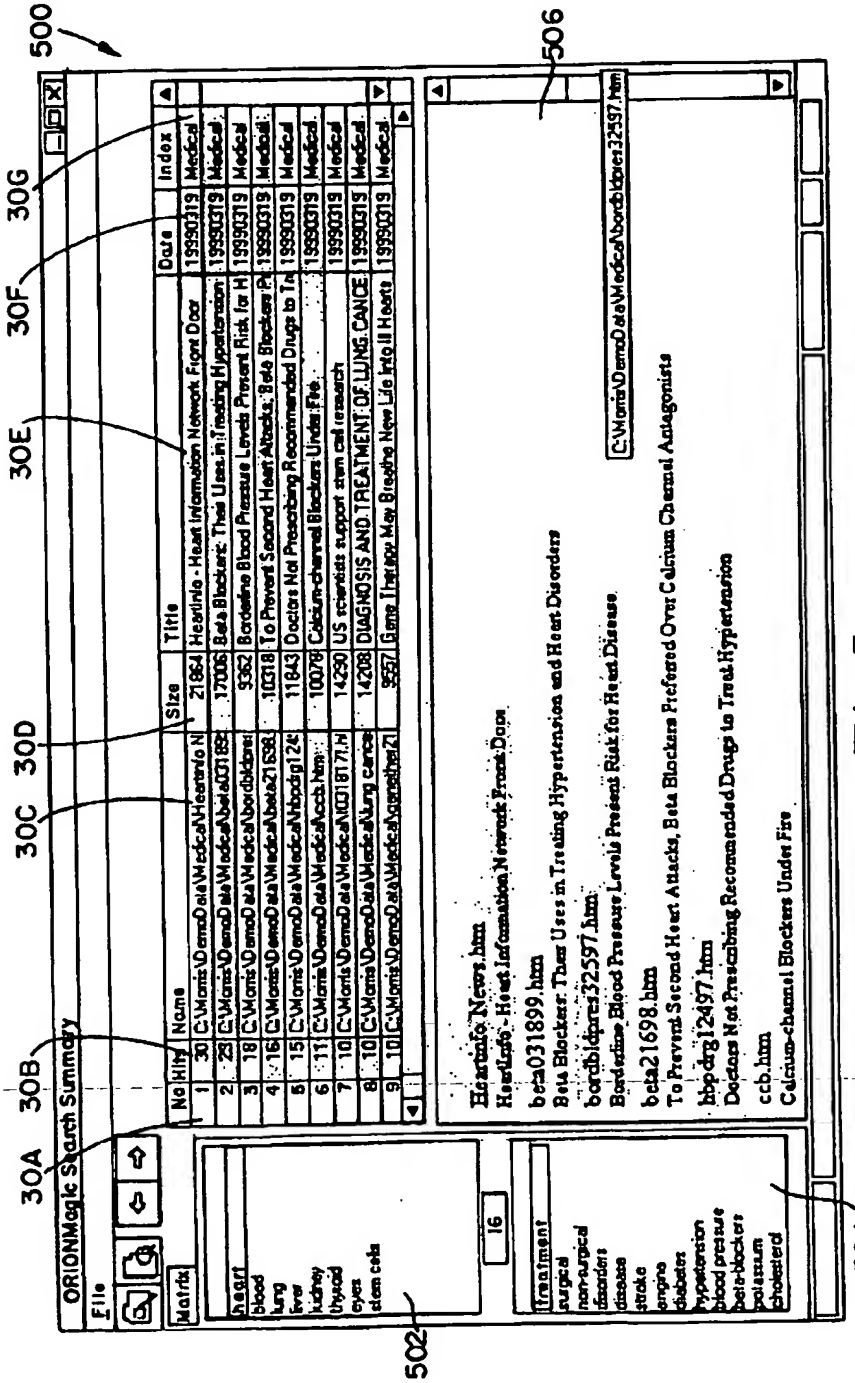

500
506
30G
30F
30E
30D
30C
30B
30A
502
504
ORIONMagic Search Summary
File
Matrix
No
Hits
Name
Size
Title
Date
Index
Medical
heart
blood
lung
liver
kidney
thyroid
eyes
stem cells
16
Treatment
surgical
non-surgical
disorders
disease
stroke
angina
diabetes
hypertension
blood pressure
beta-blockers
potassium
cholesterol
Heartinfo News.htm
Heartinfo - Heart Information Network Front Door
beta031899.htm
Beta Blockers: Their Uses in Treating Hypertension and Heart Disorders
bordpdpres32597.htm
Borderline Blood Pressure Levels Present Risk for Heart Disease.
beta21698.htm
To Prevent Second Heart Attacks, Beta Blockers Preferred Over Calcium Channel Antagonists
hbp.drg12497.htm
Doctors Not Prescribing Recommended Drugs to Treat Hypertension
ccb.htm
Calcium-channel Blockers Under Fire
C:\Morris\DemoData\Medical\bordpdpres32597.htm

Fig.5

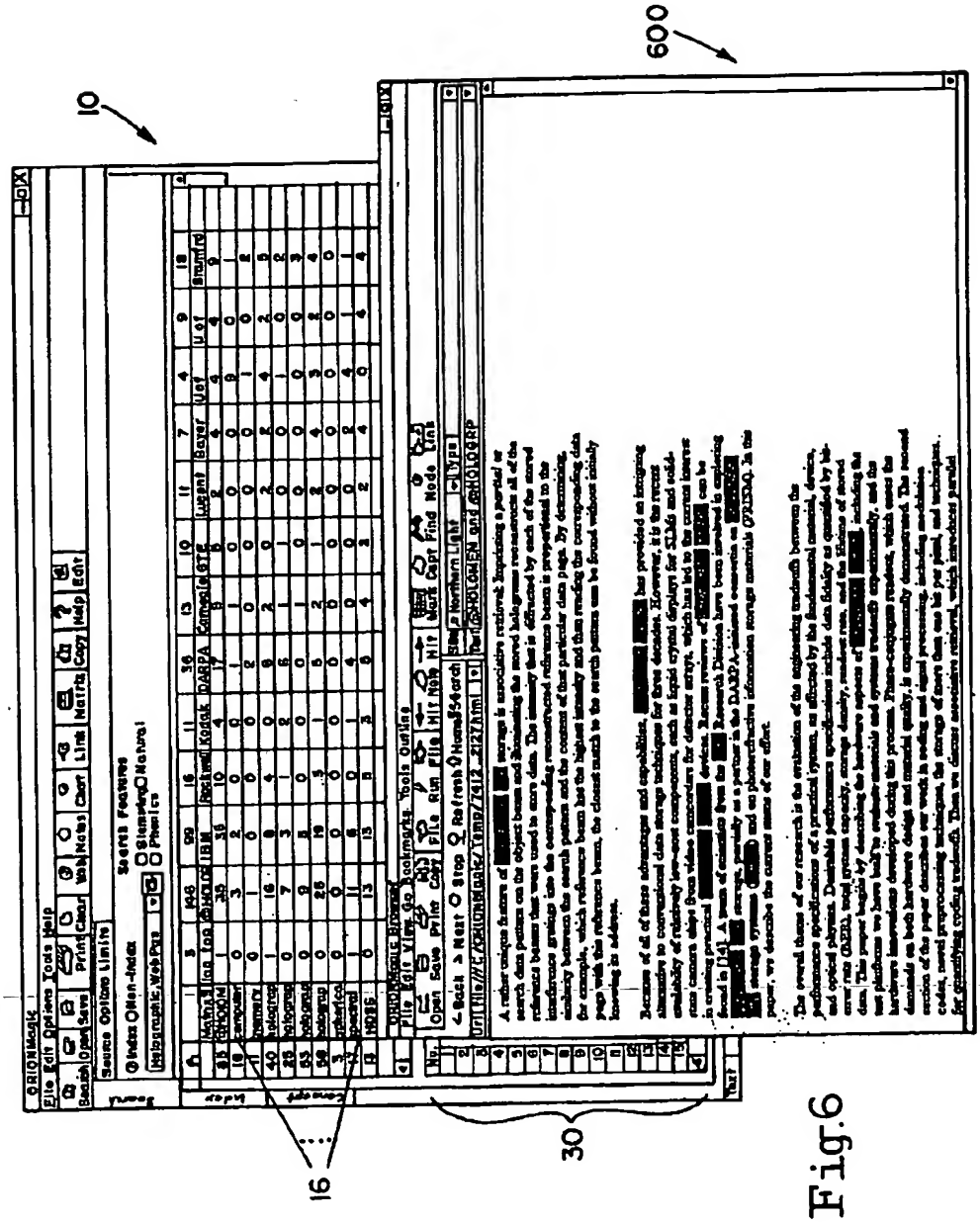


Fig.6

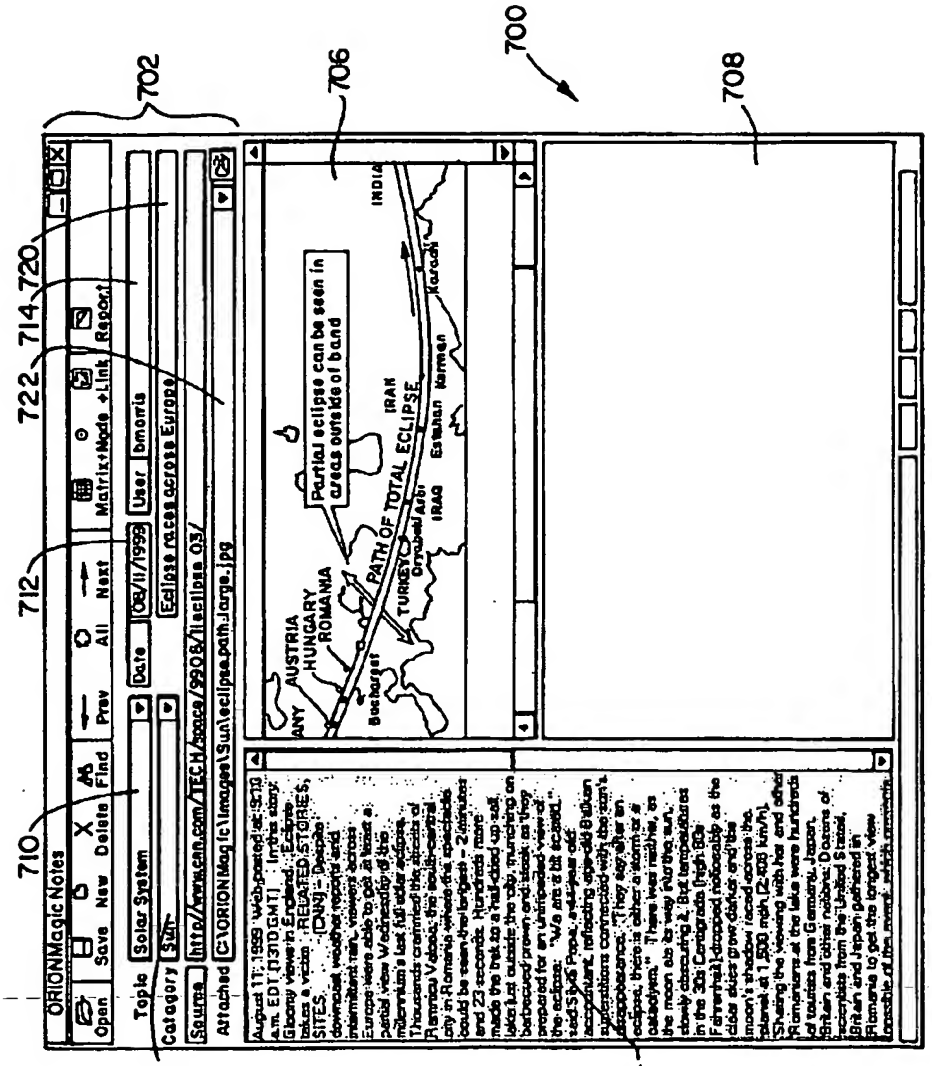

700
702
704
706
708
710
712
714
716
720
722
ORION Magic Notes
Open
Save
New
Delete
Find
Prev
All
Next
Matrix+Node
+Link
Report
Topic
Solar System
Date
08/11/1999
User
bmorris
Category
Sun
Eclipse races across Europe
Source
http://www.sun.com/TECH/space/9908/lisellipse_03/
Attached
C:\ORION\Magic\Images\Sun\eclipse.path.large.jpg
INDIA
Karachi
IRAN
Esfahan
Yerevan
IRAQ
Baghdad
TURKEY
Ankara
AUSTRIA
HUNGARY
ROMANIA
Bucharest
Partial eclipse can be seen in areas outside of band
PATH OF TOTAL ECLIPSE

Fig.7

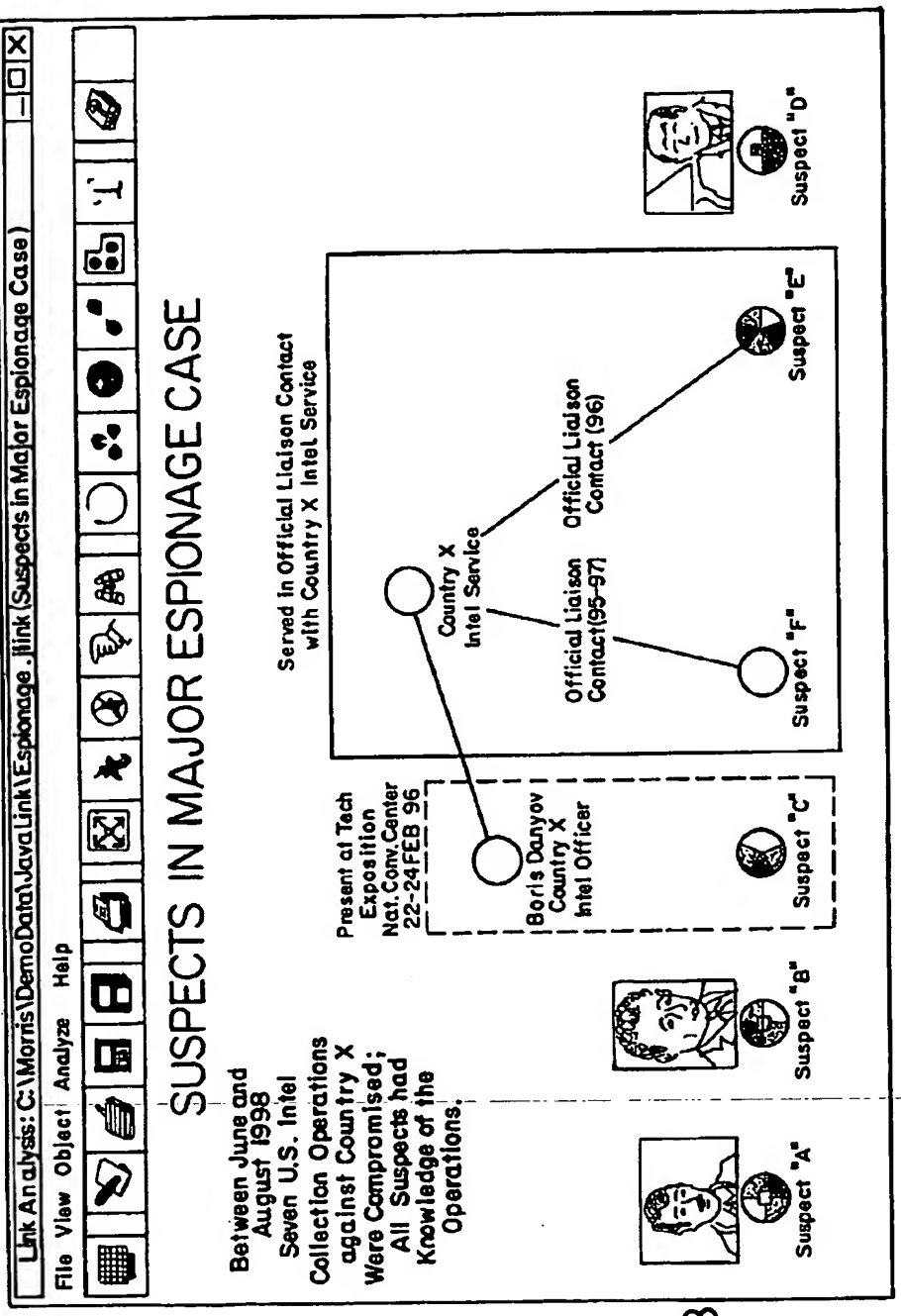

Link Analysis: C:\Morris\DemoData\JavaLink\Espionage.jlink (Suspects in Major Espionage Case)
File View Object Analyze Help
SUSPECTS IN MAJOR ESPIONAGE CASE
Between June and August 1998 Seven U.S. Intel Collection Operations against Country X Were Compromised; All Suspects had Knowledge of the Operations.
Served in Official Liaison Contact with Country X Intel Service
Present at Tech Exposition Nat.Conv.Center 22-24FEB 96
Country X Intel Service
Official Liaison Contact(95-97)
Official Liaison Contact (96)
Boris Danyov Country X Intel Officer
Suspect "A"
Suspect "B"
Suspect "C"
Suspect "D"
Suspect "E"
Suspect "F"

Fig.8

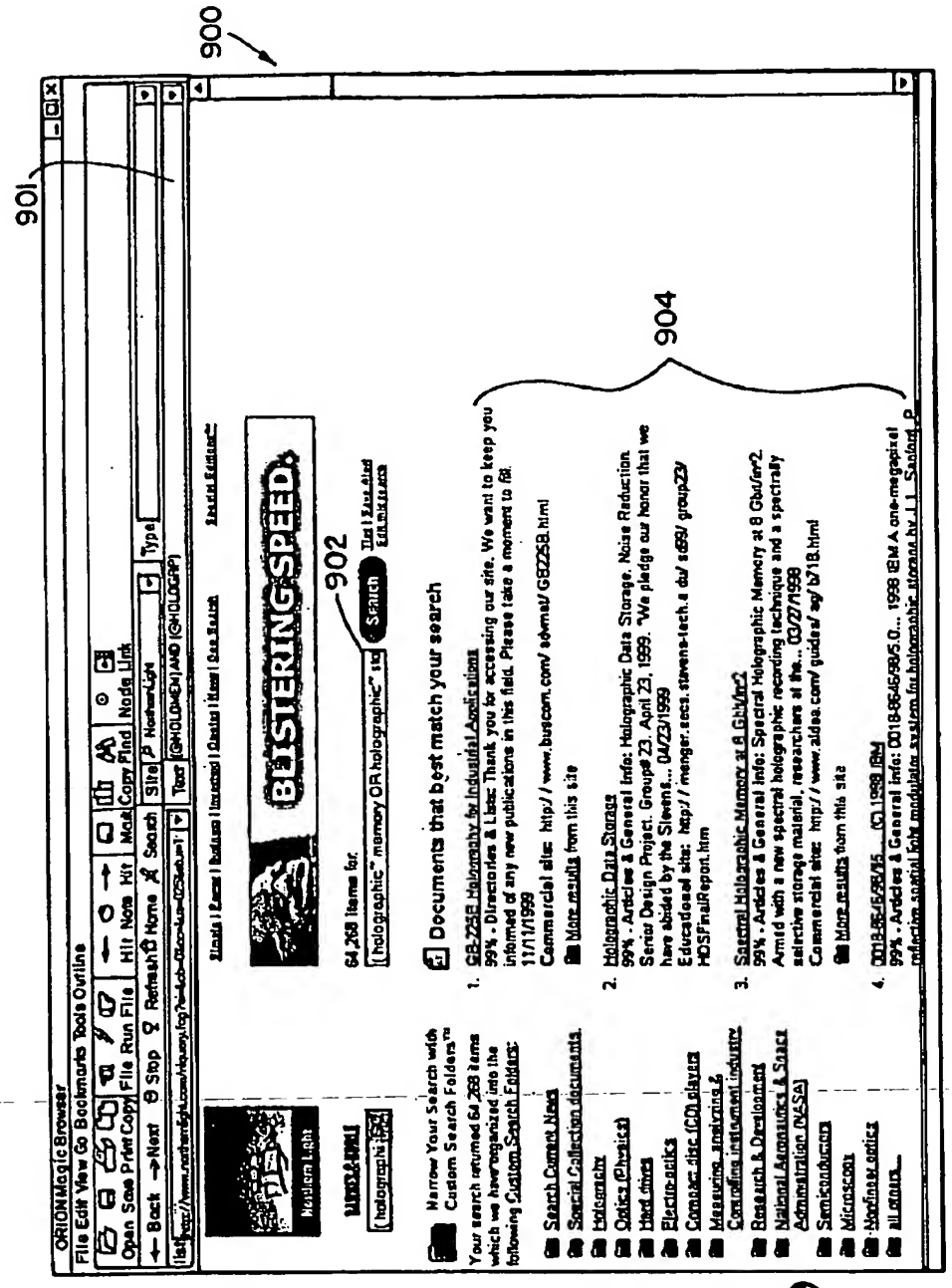

900
901
902
904
ORION Magic Browser
File Edit View Go Bookmarks Tools Outline
BLISTERING SPEED.
Documents that best match your search
Narrow Your Search with Custom Search Folders™
Search Current News
Social Collection documents
Holography
Optics (Physics)
Hard drives
Electronics
Consumer disc (CD) players
Measuring, analyzing & controlling instrument industry
Research & Development
National Aeronautics & Space Administration (NASA)
Semiconductors
Microscopy
Nonlinear optics
All folders

Fig.9

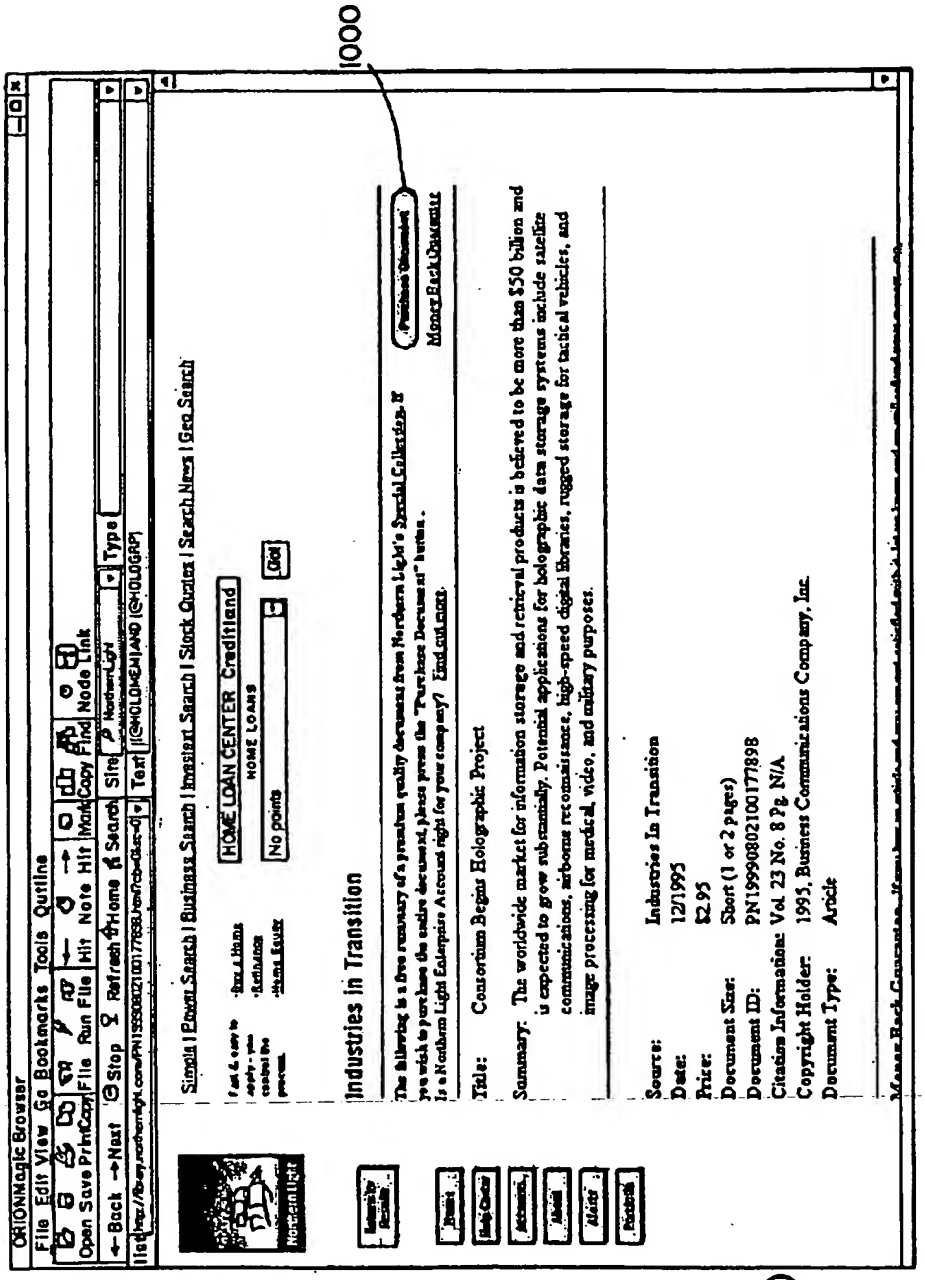

1000
ORIONMagic Browser
File Edit View Go Bookmarks Tools Qurtine
Open Save PrintCopy File Run File Hit Note Hit MarkCopy Find Node Link
Back Next Stop Refresh Home Search Site Type
Text
Simple | Event Search | Business Search | Investext Search | Stock Quotes | Search News | Geo Search
HOME LOAN CENTER Creditland
HOME LOANS
No points
Get
Industries in Transition
Consortium Begins Holographic Project
Title:
Summary: The worldwide market for information storage and retrieval products is believed to be more than $50 billion and is expected to grow substantially. Potential applications for holographic data storage systems include satellite communications, airborne reconnaissance, high-speed digital libraries, rugged storage for tactical vehicles, and image processing for medical, video, and military purposes.
Source: Industries In Transition
Date: 12/1995
Price: $2.95
Document Size: Short (1 or 2 pages)
Document ID: PN19990802100177898
Citation Information: Vol. 23 No. 8 Pg. N/A
Copyright Holder: 1995, Business Communications Company, Inc.
Document Type: Article

Fig.10

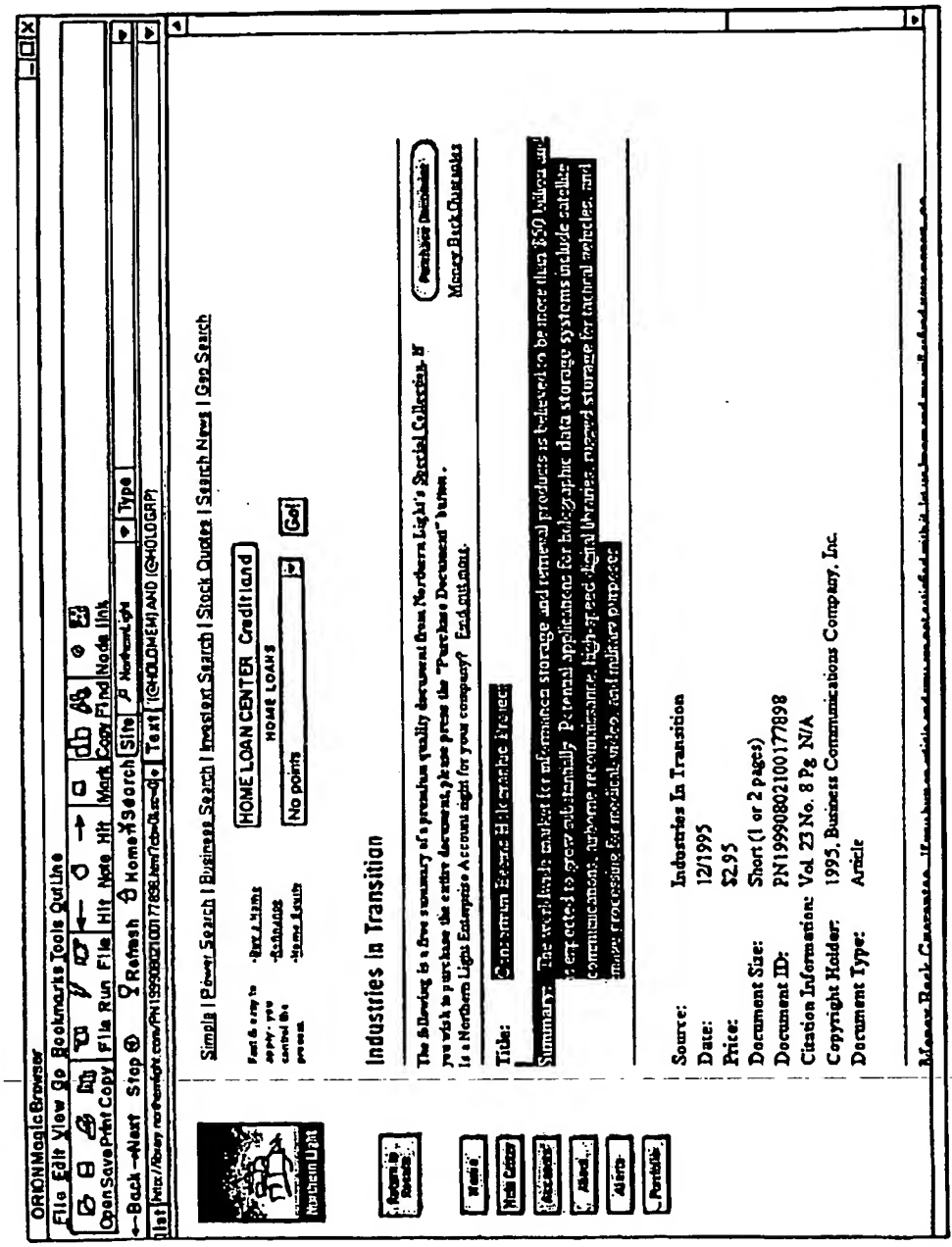

ORION Magic Browser
File Edit View Go Bookmarks Tools Outline
Open Save Print Copy File Run File Hit Note Hit Mark Copy Find Node Link
Back Next Stop Refresh Home Search Site
Simple | Power Search | Business Search | Investext Search | Stock Quotes | Search News | Geo Search
HOME LOAN CENTER Credit Land
HOME LOANS
No points
Go!
Industries In Transition
Title:
Summary:
Source: Industries In Transition
Date: 12/1995
Price: $2.95
Document Size: Short (1 or 2 pages)
Document ID: PN19990802100177898
Citation Information: Vol. 23 No. 8 Pg. N/A
Copyright Holder: 1995, Business Communications Company, Inc.
Document Type: Article

Fig.11

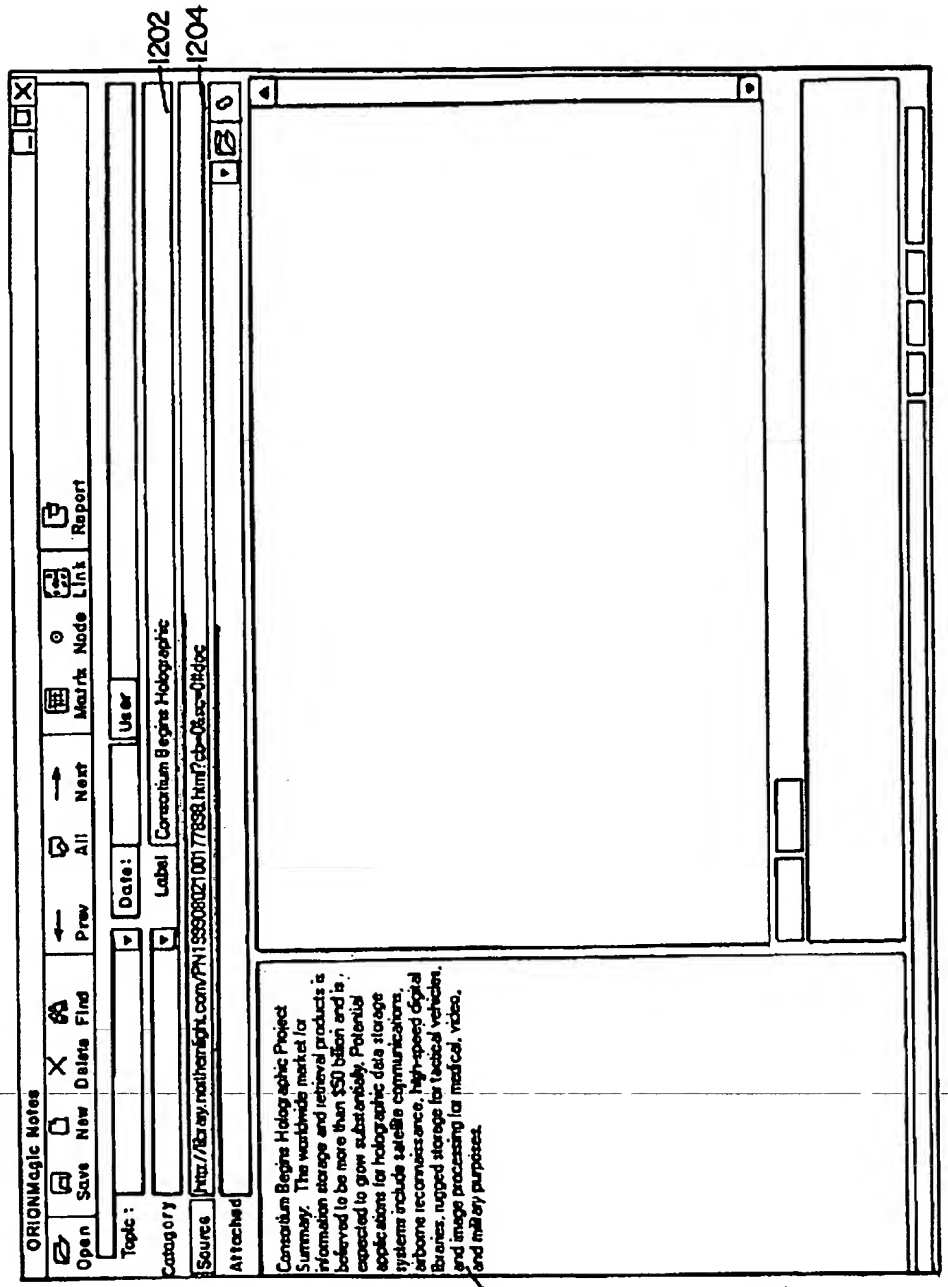

1202
1204
ORIONMagic Notes
Open
Save
New
Delete
Find
Prev
All
Next
Mark
Node
Link
Report
Topic :
Date:
User
Category
Label
Consortium Begins Holographic
Source
Attached
1203

Fig.12

METHOD AND APPARATUS FOR ELECTRONIC FILE SEARCH AND COLLECTION

U.S. Provisional Application Ser. No. 60/210,482, filed Jun. 9, 2000, is hereby incorporated by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to a method for searching electronic data files and, more particularly, to a method including the entering of a two-dimensional array of search concepts, each concept being predefined key words and expressions or user-defined key words and expressions, and detecting and displaying a correlation of occurrence, within the electronic data files, between entered concepts in the respective dimensions.

2. Related Art

The amount of information generated, collected, stored, communicated and accessible through the electronic media is continuing to increase. The increase is not only in the volume; it is in the number of sources, and the variety of formats in which the information is communicated and stored. The sources include newspapers, technical journals, government publications, literary works, laws, court opinions, business reports, and public records. More and more of these are being generated, stored, searched, retrieved, and distributed through networked systems of digital computers and other digital document generation and management devices. The migration of these and other sources, and large archives of the same, to electronic media is generally attributed to a combination of the Internet and the increasing number of and capabilities of personal computers (PCs) and other Internet access devices.

The average operator-user with an entry-level PC, a telephone line, and a subscription to an Internet Service Provider (ISP), such as America On Line®, now has access to literally billions of documents, forms, images, and text files, stored throughout the world on a myriad of databases. A large number of the databases are available as free access, to anyone, while others are subscription based or otherwise limited access. There are large databases which, although not directly accessible through the World Wide Web, are available through controlled-access wide area networks (WANs). As known to persons skilled in the relevant art, these may be physically separate from the Internet or may be Virtual Private Networks (VPNs) which coexist on the Internet with public data traffic. Through such private networks an authorized person may have access to large proprietary databases of technical journals, customer profiles, medical records, criminal records, internal memoranda, business reports and the like.

There are continuing problems, though, with searching such a large number of electronic files. Many of these problems prevent users from fully exploiting the Internet, and other wide area networks, and the many databases which these networks make available for their use. One of the problems is the formulation of a search strategy. Search strategy includes the choice of particular features that the user believes, or has otherwise determined, would be contained in, described by, or descriptive of the electronic files relating to the topic that he or she is researching. The choosing of these search features is critical to the research task, yet in most cases it is carried out using nothing more than intuition, trial and error.

Stated more particularly, a typical search of the World Wide Web is as follows: A user accesses the Internet

through, for example, an Internet Service Provider such as America On Line®. The user then, using computer software features that are well known in the art, enables a web browser program that resides on his or her personal computer, such as, for example, Microsoft Explorer® or Netscape Navigator®. As is well known in the art, the web browser is usually programmed with a default "home page", which is the Universal Resource Locator ("URL") of a specific web site. The web browser then performs the required Hypertext Transfer Protocol ("HTTP") communications with the web server hosting the home page.

The home page may be hosted by a commercial web services/advertising entity, such as Microsoft Network®, Excite®, and Yahoo®. Such commercial home pages generally have one or more icons representing search engines, both their own and those of third parties such as Lycos® and Infobot®. When the user clicks on the search engine, he or she is presented with a display page typically having a field for entering the search query terms, also referenced in the art as "key words".

The typical user then proceeds to enter the key words. Many commercially available Internet search engines provide Boolean connectors of AND, OR and NOT for connecting the key words. Boolean searching ideally identifies all documents containing the defined connection of string of "key words". This may be with or without further limitations, such as year, language, publisher, and other type characteristics. Some of the sophisticated Boolean search methods permit the user to define search terms to include not only the term itself, but also the synonyms of, and the ranges around the term. There are available search engines that have the ability to group key words according to parenthesis. This permits more complex Boolean expressions.

The entry field, though, forms the key words into a one-line expression, regardless of the number of terms. Therefore, in that one line expression, the user is attempting to formulate a single Boolean expression that will, based only on his or her intuitive sense, have a "feels OK" likelihood of finding relevant files, i.e., "hits", but is not so broad that it retrieves an unwieldy number.

In a typical scenario of Boolean searching, however, the user would not simply formulate a single expression, and then conduct the entire search using only that expression. Instead, the process is typically as follows: The user attempts a first Boolean expression and gets a number of "hits". If the number of hits is zero the user will usually vary the expression, either by removing one of the AND operators and thus lowering the criteria required for a document to qualify as a hit, or by substituting a synonym for one or more of the search terms. If the number is too high the user may retrieve, by one of the known methods, a sample set of the "hits" and read them to identify his or her next strategy. Most often the user will simply add further search criteria, typically by connecting another key word to the original Boolean phrase by an AND operator, and then run another search. When the process is completed, which is frequently coincident with the point where the user runs out of time, the typical user will have attempted a generally random sequence of different Boolean expressions, and many variations on each. The user has, hopefully at least, laboriously retrieved and reviewed documents obtained from each search expression and, in a method that is typically unique to each user, has collected and combined these into, for example, a research report.

There are numerous problems with this method. One major problem is that the user is attempting to find an

optimal search phrase, using the number of "hits" resulting from each attempt compared to the previous attempt as the sole heuristic. For example, assume that a user is writing a paper on trends in the number of children who are transported to and from school by busses as compared to the number who are transported by parents or guardians. Assume that the first Boolean phrase that the person uses is the previous example of (CHILD OR KIDS) AND (BUS OR ("PUBLIC TRANSPORTATION")). Assume that the user is searching the Internet, using known methods of Internet access. If the number of hits is too high the user will add another search term. An example would be PERCENTAGE TRANSPORTED. The typical user would then run the search again and see the number of hits. After a number of iterations the user would finally obtain an acceptable number of hits, for example thirty.

The search "methodology" described above has other shortcomings. One is that the user might not record the various search Boolean phrases that were attempted before he or she finds the phrase that yields the desired thirty hits. As a result the user might run the same search twice, or might forget to try all possible substitutions of terms. Another problem, which is more fundamental, is that the search phrase that the user ended up with might not be the only search phrase that obtains thirty hits, and, of those phrases, it might not be the best one.

Still another problem, which overlays all of the previously identified problems, is that some users are better than others at formulating search expressions. This creates a statistical variance in the "quality" of searches, both in terms of time and coverage, which may itself be a problem, especially within certain institutions and professions.

Another problem with a "methodology" for Boolean searching such as the example above is that the user may not have fully defined or developed the topic of the paper before starting the research. As is well known among, for example, college students, the user frequently starts the search before fully identifying the topic, scope, or conclusion of the task for which the search is being conducted. The user then picks the topic, and composes the outline of the paper, or other reporting document, after sifting through the results obtained from his or her repeated searches with different Boolean expressions. However, in using the "trial and error" method of attempting numerous Boolean expressions to see which one provides results that inspire the user, the user may frequently overlook many Boolean expressions for which the search results would reveal more interesting or valuable topics.

Yet another problem with the prior art of searching using single-line Boolean expressions is that many users cannot easily generate or store an understandable description of, or history of, the overall search strategies that were employed when he or she conducted a search. Therefore, frequently the user will run what is basically the same search twice, or will recreate the search strategy each time a particular project is picked up again or a new research project is undertaken.

Still another problem is that after trying multiple Boolean phrases and obtaining and relying on the results obtained with one or more of the searches, the user may have difficulty ascertaining or defending the quality of the search. This is the problem that may be encountered by students, as well as consultants and analysts when having to defend the facts, analysis or conclusion presented in a final paper based on research results.

SUMMARY OF THE INVENTION

The present invention provides a structured, concept-exhaustive method for searching databases for documents and other electronic files by receiving a plurality of search concepts from the user, designating a first plurality of the search concepts as a first search vector defining a first dimension of the matrix, and designating a second plurality of the search concepts as a second search vector defining a second dimension of the matrix. The method then performs a search of one or more databases based on the matrix, and identifies a plurality of search results, each represented by a cell of the matrix. A row of the matrix is formed by a row of cells reflecting, on a one to one basis, a search result for each of the plurality of search concepts within the first search vector. A column of the matrix is formed by a column of cells reflecting, on a one to one basis, a search result for each of the plurality of search concepts within the second search vector. Other cells of the matrix reflect, on a one to one basis, a search result for each unique pair comprising a search concept from among said first plurality of search concepts and a search concept from among said second plurality of search concepts.

A further embodiment of the invention presents the user with a visual display arranging the first plurality of search concepts as a border column, and the second plurality of search concepts as a border row. Each cell within the border is in a row-column position corresponding to a pair of search concepts, one being from the first plurality of search concepts and one being from the second plurality of search concepts. The step of displaying the search results forms each cell to have a visual state reflecting the search result for the search concept or pair of search concepts corresponding to that cell.

A still further embodiment of the invention includes a step of displaying the matrix of cells to appear as a two-dimensional plane, and displaying the search results to appear as a third dimension.

A further embodiment of the invention includes a step of receiving a search concept definition command from a user, and defining one or more of the plurality of search concepts in accordance with the received search concept definition command.

Another embodiment of the invention may be combined with any of the previously identified embodiments, and comprises the further step of receiving a user-entered cell selection command, presenting the user with a cell result list identifying documents and other electronic files within the search results reflected by the selected cell. This embodiment optionally includes a further step of receiving a document selection command from the user and a step of displaying information reflecting information content of a document or other electronic file selected in accordance with the document selection command. This embodiment optionally includes a further feature of simultaneously displaying the received cell selection command, the cell result list, a data reflecting the document selection command, and the information reflecting information content.

A further embodiment of the invention may be combined with any of the previously defined embodiments of matrix searching in accordance with the present invention, and includes the further steps of receiving a collection document command from the user, and generating a collection document in response, receiving a document selection command from the user, displaying a document or other electronic file in response, receiving a portion storage command, and copying of information into the collection document from a portion of the displayed document corresponding to the portion storage command.

A still further embodiment of the invention includes an organizing step which may be combined with any of the

previously defined matrix searching with collection embodiments, and includes the further steps of receiving a user-entered document tag data, and storing an information into the collection document corresponding to the received document tag data and a portion of the displayed document corresponding to the portion storage command. An optional feature of this embodiment includes a user-entered relational database information data with the document tag data. A further optional feature of this embodiment includes steps of receiving a collection document store command from the user, and storing the collection document into a collection database in response, and repeating the step of matrix searching to including searching the collection database.

A further embodiment of the invention includes a reporting step which may be combined with any of the previously defined matrix searching with collection and organizing embodiments, and includes the further steps of receiving a user-entered link analysis generation command, identifying information contained in the search result documents that is common between two or more search concepts, and generating a link document having a link information reflecting the information identified as common. An optional feature of this embodiment includes a step of generating a graphical link chart showing the link information.

A still further embodiment of this invention comprises any of the previously defined embodiments combined with a step of drill down matrix searching, the drill down matrix searching comprising the step of receiving a cell search command from the user, receiving a new plurality of search concepts from the user, the receiving including entering or designating a first plurality of the new search concepts as a first search vector defining a first dimension of a new matrix, and entering or designating a second plurality of the new search concepts as a second search vector defining a second dimension of the new matrix. This embodiment then searches, based on the new matrix, the documents and other electronic files represented in the search results within a cell corresponding to the received cell search command.

These and other objects, features and advantages of the present invention will become more apparent to, and better understood by, those skilled in the relevant art from the following more detailed description of the preferred embodiments of the invention taken with reference to the accompanying drawings, in which like features are identified by like reference numerals.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is an example high level functional flow chart of a method according to the present invention;

FIG. **2** shows an example graphical user interface display of a search matrix in accordance with the method of the present invention;

FIG. **3** shows an example graphical user interface of an indexing step and concept definition step in accordance with the method of the present invention;

FIG. **4** shows an example of three-dimensional graphical search matrix display in accordance with the method of the present invention;

FIG. **5** shows an example display of a search result speed reading feature from the graphical user interface of FIG. **2**;

FIG. **6** shows an example of the search graphical user interface according to FIG. **2**, using different search queries, with an overlay of a jump window using a browser feature of the present invention;

FIG. **7** shows an example graphical user interface for a multimedia search information collection step in accordance with the method of the present invention;

FIG. **8** shows an example display of a search result linking feature in accordance with the method of the present invention;

FIG. **9** shows an example graphical user interface for searching the same queries as FIG. **8**, using a different search database selection, and linking to a third party search engine;

FIG. **10** shows an example display of a hyperlink accessed through the example graphical user interface display of FIG. **9**;

FIG. **11** shows an example of a user collecting information from the example graphical user interface of FIG. **10**; and

FIG. **12** shows an example of a search collection document into which the information highlighted in FIG. **11**, and related user-generated tags, are inserted.

DETAILED DESCRIPTION OF THE INVENTION

A method for a structured, but flexible search using an integrated graphical user interface for entering a user-specified matrix of both pre-defined and user-defined search concepts, performing a complete and exhaustive search, presenting the user with a comprehensive overview of the search statistics corresponding to the matrix of concepts, and with information from documents relating to selected search concept, and quickly linking the user to the search documents from the integrated graphical user will be described. Also described are additional features and embodiments, including analyzing the results of the search in relation to the search matrix, collecting the search results with assistance of a display of the search matrix, organizing the collected search results by receiving tag and relational database data from the user and merging that data with the collected search results, and generating a link report identifying and showing linkages between search concepts reflected by information in the search results.

The following description includes numerous example details and specifics which pertain only to the specific examples presented herein, and are included only to assist in describing these specific examples, and thus assist the reader in understanding through example the features and elements of the present invention. It will be evident to ones skilled in the art that the invention can be practiced without, and with different ones of, these details and specifics. Further, this description assumes an ordinary skill in the art of conventional Boolean and other commercially available search engines, known, standard network and database structures and interface protocols, and conventional programming in, for example, Visual Basic®, C++, or Java®, running under, for example, Microsoft Windows 95®, Microsoft Windows NT®, Linux, Sun Solaris®, or Apple OSX®, or other equivalent commercially available operating systems.

FIG. **1** shows a high level functional flow chart of a preferred embodiment of the invention. It will be understood that FIG. **1** is not a limitation on the invention, and that the function and novelty of the invention does not depend on all of depicted functional blocks being present. Instead, the invention can be practiced and its benefits over the existing art obtained using only a subset of the depicted number of functional blocks, as will be better understood upon reading this description in view of the accompanying drawings. Further, the particular segmentation and labeling of FIG. **1** is not a limitation on the particular software coding scheme or software module structure for implementing the invention. As will be understood by persons of ordinary skill in the arts relating to this invention, the described invention can be

readily implemented with a segmentation, arrangement and labeling of functional blocks differing from the example shown in FIG. **1**.

Referring to the example flow chart of FIG. **1**, user first selects, at block **100**, one or more global data sources "GDS". Global data sources, or GDS, is defined herein as the universe of databases which the user wishes to search. In the example described here, step **100** is where the user selects the database that he or she will search. It is assumed that a system setup has made the databases available to the user, according to computer and database set-up procedures well known to persons of ordinary skill in the art relating to this invention. For example, if the World Wide Web, or other Internet-related database resource or networked database is among those that can be selected by the user, then the user's computer (not shown) will typically have an Internet web browser, such as Microsoft Internet Explorer®, and the user's computer will typically be connected via modem to an Internet Service Provider, such as America On Line.

For purposes of this description, "database" means any stored collection or aggregation of digital information, which may be arranged in and according to any of the various formats known in the art including, for example, records, tables, word processing documents, text documents, sound files, and image files. These digital information formats are collectively referenced herein as "electronic files". The databases selected at step **100** may be in accordance with any of the structures known in the art. It is assumed that the person of ordinary skill in the arts relating to this invention has a working knowledge of the available database structures and, therefore, a detailed description of database theory is not necessary for an understanding of this invention.

For purposes of example, the World Wide Web is one of the databases GDS which may be selected at step **100**. As known in the art, the "database" embodied in the World Wide Web comprises more than one billion electronic files, which are "posted" by storing them within one or more web servers (not shown) that are connected to and accessible through the Internet. Each posted file is addressed, and accessed, by its Uniform Resource Locator (URL) value. Electronic files posted in this manner are generally referenced as "web documents". Business entities such as Google® and Yahoo® maintain an index of some, but not all, of the publicly available web documents. As will be described in more detail with respect to the indexing step **102**, some of the web documents' index entries include key words and other file description data characterizing the web document.

For purposes of example, another type of database known in the art which may be selected at step **100** for searching stores electronic files in a name table having, for each file, addressable locations within a storage media identifying where the file is stored. Example storage media includes a magnetic disc storage unit, an optical disc storage unit, a solid state storage unit, or networks thereof. Commercially available products, methods and protocols of this type of database are well known in the art.

Still another type of database which may be searched in accordance with this invention, and thus selected at step **100**, is the "relational database". The term "relational database" is defined herein in accordance with its established meaning in the relevant art. An example of a relational database is Microsoft Access®.

The GDS databases selected at step **100** may include publicly accessible web documents which, although posted, do not have their URLs listed by well-known entities such as Yahoo® and Google®. Further, the selected GDS databases may be databases for which the documents themselves are not "posted" but, instead, have a posted web document which is an index of that database. As known in the art, search engines may access such an index, either directly or indirectly through the indexes posted by entities such as Yahoo® and Google®, and obtain "hits" referencing the user to the home page of the database. Typically, as known in the art, the user then accesses the specific documents using another search engine featured on that home page.

Still other databases selected at step **100** may include, depending on particular system access privileges that the user possesses, controlled access databases owned or controlled, for example, by business entities such as banks, insurance companies, and medical institutions, or by government entities. The specific structures and protocols of such controlled access databases, as well as the hardware and software resources for maintaining them, are known to one of ordinary skill in the arts relating to this invention, can be readily interfaced with the method of the present invention, and therefore description is omitted.

The particular user interface by which the user selects databases at step **100** is a design choice readily implemented by one of ordinary skill in the arts pertaining to this invention. An example is shown in FIG. **2**, which is an example graphical user interface for performing the search step **104** described below. Referring to FIG. **2**, the depicted example graphical user interface **10** has a data entry field **12** into which the user enters the name of the GDS databases selected at step **100**. As discussed in reference to the indexing step **102** below, the user can enter an index name into field **12** instead of a database name. In fact, the example search shown in FIG. **3** searches an index, which is named "Medical".

Referring again to FIG. **2**, a further option in the graphical user interface **10** is a field **14** for the user to specify a higher level domain in which the GDS database is found. The FIG. **2** example graphical user interface **10** shows the field **14** having a domain labeled "local". Referring to the FIG. **2** example, the "local" domain identifier in field **14** informs the user's database access computer (not shown) that the "Medical" database index is local to the computer. As known to ones skilled in the relevant arts, the meaning of "local" is determined by the particular computer and operating system on which the invention is being practiced. For example, as known to ones skilled in the relevant arts, within a computer system running under the Windows 95® or NT® operating system, "local" can be defined as one or more hard drives (not shown) or other storage media connected directly to the computer, or shared by multiple computers connected by a LAN (not shown) to one or more servers (not shown).

It will be understood that a plurality of GDS names or identifiers may be entered into, for example, field **12** of the graphical user interface **10** of FIG. **2**.

It will also be understood that step **100** may be omitted, and the invention practiced with, or with the user being limited to, default databases. Further, it will be understood that step **100** may be an automatic selection process, using a computer program readily composed by one of ordinary skill in the art for selecting one or more GDS databases in accordance with the subject matter searched.

After the user selects the GDS databases for searching at step **100**, or as an optional first step if the selection step **100** is omitted, the user may employ step **102** to pre-index the databases to be searched. The pre-indexing step **102** is not

required for practicing the method of this invention. However, as known in the relevant arts, query searching for files meeting a criteria is typically much faster if it is performed, at least in part, on an index of the files instead of the actual content. The general reasons are well known, and include the fact that, typically, index files are typically much shorter than the full text files that they characterize, and that index files are typically formatted so that specific information is in specific fields. However, as is also known in the art, there are typical shortcomings with index-based searching. These include the fact that the ultimate coverage and accuracy of the search depends, in part, on the accuracy of the index.

The specific types and methods of indexing performed at step **102** are in accordance with any of the indexing schemes known in the relevant arts. Step **102** may, for example, process each file or document in the selected databases GDS and generate a key word or key feature profile record corresponding to that file. As known in the art, the World Wide Web uses this type of indexing. More particularly, entities such as Google® and Yahoo®, maintain indexes based on the web documents being in the Hypertext Markup Language (HTML) format, which has metatags within the document itself describing its features. The indexes maintained by entities such as Google® and Yahoo® typically correspond to, or contain, these HTML metatags of the title, summary description, and key words. Further, the file description data may be provided by the person or business entity that owns the web document. In addition, services such as Google® and Yahoo® may send, typically via the Internet, a file extraction program, frequently referenced in the art as a "spider", to extract additional data for creating the index.

Notwithstanding the indexes maintained by business entities such as Yahoo® and Google®, step **102** may index the World Wide Web using the above methods. Methods for generating indexes of the World Wide Web are well known in the art and are therefore not described here. As known in the art, such indexing is a substantial task requiring considerable resources, although there may be reasons for step **102** to generate such an index. Consideration of these must, as known to persons skilled in the art, be in view of the particular objectives and resources of the user. For example, it is well known that many commercial World Wide Web indexes are typically incomplete and have inconsistent accuracy. In addition, such indexes are generally formatted for interfacing with the entities' own search query engines. Therefore step **102** may include generation of a customized World Wide Web index, additional to those available through Yahoo®, Google® or even Altavista®.

Typically, however, the GDS databases that the user would select for indexing at step **102** would be ones other than the World Wide Web. There are numerous indexing schemes, methods, and numerous products for performing the same, which are well known in the arts relating to this invention. The specific indexing schemes and methods employed at step **102** would, as is known to persons skilled in the art of database management, is a design choice and would be selected and/or written based on factors including the storage capacity of the processing resources hosting the database(s) and the search performance desired by the user. One example that is widely available is the commercial database management and indexing program sold as Veritas®.

Referring to FIG. **3**, in a preferred embodiment of the invention the indexing step **102** is carried out by generating a graphical user interface display such as the depicted window labeled INDEX having a field area **300**. The INDEX field area **300** is for designating the indexing operations including, for example, Create New Index field **302**, Add Files to Index **304**, Re-index **306**, and Compress Index **308**. The FIG. **3** INDEX graphical user interface further includes a field **310** for the user to enter a name for the index, a field **312** for the user to enter the name of the directories to be indexed, a field **314** for the user to enter the names of the folders within the director(ies) entered in the field **312** to be indexed, and a field **316** for the user to enter the names of folders to be excluded from the indexing operation.

The FIG. **3** graphical user interface **300** is for purposes of example only, as the design of graphical user interfaces for selecting the files or databases to be indexed is a design choice, readily constructed by one of ordinary skill in the art of computer programming. One typical graphical user interface for an indexing that could be incorporated and used at step **102** is that shown by the Microsoft Explorer® feature of the Windows 95® operating system.

As known to persons skilled in the arts relating to this invention, database indexes such as those generated at step **102** are typically not updated each time that the indexed databases are searched. Depending on the size of the database, and the level of the index, such updating may require substantial time, and may render the database inaccessible or lessen the access performance while the process is ongoing. The database index is therefore, typically updated periodically, or in response to a specific command entered by the user. The criteria for determining the frequency of updating the index are known to persons skilled in the arts relating to this invention and, therefore, description is omitted.

Referring to FIG. **2**, which is an example graphical user interface for performing the search step **104** described below, the indexing step **102** stores the index in a file (not shown), and the user can scroll through this as shown in the FIG. **2** field **12**. The specific example index shown in field **12** of FIG. **2** is named "Medical".

As stated above, the indexing step **102** is not required for practicing this invention. The search step **104** can search indexes created by others, such as the Yahoo®, Google® and Altavista® indexes. Search step **104** can search non-indexed GDS databases. Search of non-indexed databases is performed in a manner comparable, in part, to the methods used by existing query-based search engines to search non-indexed files. More specifically, as known in the art, such searches typically examine database files one at a time, scanning for the query search terms. As is also known in the art, such searches typically require considerably greater time than indexed searches.

Referring to the example flow chart of FIG. **1**, after selecting a GDS database at step **100** and indexing one or more of the selected databases at step **102**, or as a first step if these steps are omitted or established as a default, the method of this invention performs the search step **104**. As will be described, search step **104** comprises receiving a plurality of user-entered search query terms, which are labeled for reference as CONCEPTS, and assigning or arranging the CONCEPTS into two sub-pluralities, which will be referenced as ROW CONCEPTS and COLUMN CONCEPTS. As will be further described, the CONCEPTS search query terms may be pre-defined, such as words of the English language, or may be defined by the user as will be described in greater detail below.

Referring to the example graphical user interface shown in FIG. **2**, step **104** arranges or assigns a first plurality of the

received CONCEPTS into a first set of M CONCEPTS, which are labeled herein for reference as ROW CONCEPTS (i), the index "i" ranging from i=1 to M, with an example set shown in field **16** of the figure. The step further arranges or assigns a second plurality of the received CONCEPTS into a second set of N CONCEPTS, which are labeled herein for reference as COLUMN CONCEPTS (j), the index "j" ranging from j=1 to N, with examples shown in field **18** of the figure. The arrangement or assignment of CONCEPTS as ROW CONCEPTS and COLUMN CONCEPTS is by user choice.

It will be understood that the above reference labels of COLUMN CONCEPTS and ROW CONCEPTS, their respective indices "i" and "j", and the population labels of "M" and "N" are merely for consistency of reference in describing the method of this invention. These labels and indices are not a limitation on the present method, as persons of ordinary skill in the computer arts relating to this invention can readily identify, upon reading the instant description, many alternative label and indexing schemes for practicing the inventive method described herein.

Referring to the example graphical user interface shown in FIG. **2**, after the user has entered the desired M+N CONCEPTs, in two sets arranged, for example, as the ROW CONCEPTs(i), i=1 to M, and COLUMN CONCEPTs(j), j=1 to N, the user clicks on field **20**, which is labeled "Search". In response, step **104** performs M query searches, one for each ROW CONCEPT(i), of the GDS database or database index identified in field **12** and, for each search, identifies all data files having a predetermined type of occurrence of a word or phrase that is within the definition of that ROW CONCEPT. For purposes of this example "predetermined type of occurrence" means an occurrence anywhere in the document. Step **104** also performs N query searches, one for each COLUMN CONCEPT(j) and identifies all such data files having an occurrence of a word or phrase that is within the definition of that COLUMN CONCEPT.

For each ROW CONCEPT(i) and COLUMN CONCEPT (j) searched, step **104** generates a HITS record, labeled for reference herein as HITS(ROW CONCEPT(i)) and HITS (COLUMN CONCEPT(j)). The HITS records include a K number, which equals the number of documents in the record. The K number is referenced herein as K(ROW CONCEPT(i)) and K(COLUMN CONCEPT(j)). The labels of ROW CONCEPT and COLUMN CONCEPT are for purposes of reference only, as each is a CONCEPT as defined herein. As will be understood, the assignment of ROW CONCEPTS and COLUMN CONCEPTS is a label for describing operation of separating the user-input CONCEPTS into two groups, searching each member of each group individually, and then forming all possible pairs of ROW CONCEPT with a COLUMN CONCEPT, the pair being referenced as a PAIR CONCEPT(i,j), i=1 to M, j=1 to N, and searching each pair.

Referring to the example graphical user interface shown in FIG. **2**, the hit count K(ROW CONCEPT(i) for each of the ROW CONCEPTS is displayed in a ROW CELL(i) located, for example, within an ascending ordered vertical column in field labeled **22**, which is located, in the example, to the left of the column of ROW CONCEPTS in field **16**. Likewise, the hit count K(COLUMN CONCEPT(j) for each of the COLUMN CONCEPTS is displayed in a horizontal COLUMN CELL(i), which is located, in the example, in field **24** above the row of COLUMN CONCEPTS in field **18**.

In a preferred embodiment, step **104** downloads each matching document into a storage (not shown) that is local to the general purpose computer with database access capability (not shown) on which the user is interfacing with this method through, for example, the graphical user interface of FIG. **2**. The search step **1004** thus generates records HITS (ROW CONCEPT(i)) and HITS(COLUMN CONCEPT(j)) having information uniquely identifying each document found in the GDS database with information matching the referenced CONCEPT, and automatically downloads the identified documents into the HITS records. For example, if the selected GDS database is the World Wide Web and step **104** identifies, for a particular ROW CONCEPT(i), twenty web pages having matching information, the step **104** forms the record HIT(ROW CONCEPT(i)) as a list of twenty URL's, and other information as described below, and a downloaded copy of each of the twenty web pages.

Referring to the specific example in FIG. **2**, ROW CONCEPT(**1**) is "heart", and ROW CELL(**1**) displays "15", meaning that K("heart") is "15" and, therefore, fifteen documents or other electronic files within the database being searched, which in this example is "Medical", contain a word or term within the definition of the CONCEPT of "heart". ROW CONCEPT(**7**) of the FIG. **2** example is "eyes" and ROW CELL(**7**) in field **16** displays "1", meaning that K(eyes) is "1" and, therefore, the GDS database had only one document or file meeting the definition of the CONCEPT "eye". Similarly, COLUMN CONCEPT(**9**) of the FIG. **2** example is "hypertension" and COLUMN CELL (**9**) in field **24** displays "8", meaning that K("hypertension") is "8" and, therefore, the GDS database had eight documents or files meeting the definition of the CONCEPT "hypertension".

Next, step **104** performs a systematic search of the GDS database using every possible pair of a ROW CONCEPT and a COLUMN CONCEPT. The specific order of the automatic pair formation and searching is a design choice. For purposes of example, the pair formation process begins by selecting the first ROW CONCEPT, which is ROW CONCEPT(**1**), logically pairing it, sequentially, with each individual COLUMN CONCEPT(j), j=1 to N, and, for each pair, performing a "search" of the selected GDS databases. Each pair is referenced herein as PAIR CONCEPT(**1**, j), j=1 to N. The process then selects ROW CONCEPT(**2**) and logically pairs it with each of the COLUMN CONCEPTS, to form N new PAIR CONCEPTs(**2**, j), j=1 to N. The step also searches the GDS database using each of the PAIR CONCEPTS. The process repeats until it has selected the M^{th} ROW CONCEPT, paired it with each of the N COLUMN CONCEPTS, and searched the GDS database using each. The logical operation for the pairing may be selectable but, preferably, it is the Boolean AND function. Therefore, assuming that the AND function is used, PAIR CONCEPT (**1**, **1**) is ROW CONCEPT(**1**) AND COLUMN CONCEPT (**1**).

Referring to the specific FIG. **2** example graphical user interface, the step **104** search of the GDS database using each of the PAIR CONCEPTs(i, j), i=1 to M, j=1 to N, generates a (MxN) records, labeled for reference as HITS (PAIR CONCEPT(i, j)), each identical in structure to the above-described HITS(ROW CONCEPT(i)) and HITS (COLUMN CONCEPT(j)) records. The step displays the hit count for each, referenced herein as K(PAIR CONCEPT(i, j)), in a corresponding PAIR CELL(i, j) in field **28** of FIG. **2**.

The search query operation performed for each CONCEPT(i, j) is identified as a "search", but it is contemplated that, in some applications, it may be unnecessary to have an additional interface with, or query of, the selected

GDS databases. Instead, the "search" for each PAIR CONCEPT(i, j) could compare HITS(ROW CONCEPT(i)) with HITS(COLUMN CONCEPT(j)) and identify all documents or files appearing in both HITS records. In a preferred embodiment, though, step **104** perform an actual query-based search on the GDS database using each PAIR CONCEPT(i, j). Searching each PAIR CONCEPT(i, j) may be preferable if, for example, the user had entered a limit (not shown) on the number of HITS for the ROW CONCEPTS and/or COLUMN CONCEPTS. For example, if a user or particular embodiments placed a limit of one hundred documents in each HITS record, and the GDS database were the World Wide Web then the number of documents listed in the HITS records might be a small percentage of the matching documents. If, for example, ROW CONCEPT(**1**) is "automobile", and COLUMN CONCEPT(1) is "Spanish" then the number of hits, i.e., K(automobile) and K(Spanish), is likely to be in the tens of thousands. The documents in the record HITS(Spanish), and HITS(automobile) would have only hundred of these. A comparison of the respective records of HITS(Spanish) with HITS(automobile) would, therefore, have a significant probability of missing many documents which, although having both "automobile" and "Spanish", were omitted from one or both of the one hundred documents listed in each of the HITS records. If, however, step **104** performed an actual query based search of the World Wide Web using the paired CONCEPT (automobile, Spanish) it is likely that the search would return a usable quantity of matching documents.

The above-described search step **104** may include a filter that is additional to the criteria set by the CONCEPTS. The filter may qualify documents based on features including, but not limited to, date, source, author, format and size. For purposes of this example the type of occurrence is anywhere in the document.

The specific code-level process of searching the database in field **12** for occurrences of each ROW CONCEPT(i), COLUMN CONCEPT(j) and PAIR CONCEPT(i, j) is a design choice, readily made by one of ordinary skill in the art. As known in the art, the specific code-level process depends, in part, on the structure of the database searched, whether or not the search is index-based, and on the interface requirements particular to the commercial database management system (DBMS) on which the database in field **12** is structured.

Referring to the FIG. **2** example graphical user interface, PAIR CONCEPT(**1, 1**) as identified above is the CONCEPT "heart" ANDed with the CONCEPT "treatment". This is referenced herein as PAIR CONCEPT(heart, treatment). As shown in FIG. **2**, PAIR CELL (**1, 1**) displays "14", meaning that fourteen documents or files within the GDS database had information meeting the definition of "heart" and meeting the definition of "treatment". Similarly, PAIR CONCEPT(**4, 6**) is ("liver","stroke"). It is seen from the "0" displayed in PAIR CELL(**4, 6**) that K("liver", "stroke") is "0", meaning that the "Medical" index of this example did not contain any documents or files having information meeting both the definition of the CONCEPT "liver" and the CONCEPT "stroke".

As described below, the scope of information that is within the meaning of a CONCEPT is determined by the user-entered, or previously stored Boolean phrase defining the CONCEPT, and by EXPANDORS added by the user to the typed letter strings appearing in fields **14** and **18**.

As described and shown by the FIG. **2** example graphical user interface, the method of this invention provides the user, by clicking on the "Search" button **20** one time, with the results of M plus N, plus (M times N) searches. The user then sees in, for example, the graphical user interface of FIG. **2**, the comparative frequency of occurrence of all M of the ROW CONCEPTS, all N of the COLUMN CONCEPTS, and the (MxN) pairings formed by PAIR CONCEPTS(i, j), for i=1 to M, j=1 to N. This is substantially more efficient, and more exhaustive in its search coverage, than the prior art method of picking a search query term, or formulating a single Boolean expression, performing a search, observing the results, and then picking a succession of other terms and expressions until the result "looks right" to the user.

For example, in the specific example depicted in FIG. **2** the number of ROW CONCEPTS is seven and the number of COLUMN CONCEPTS is twelve. Step **104** then generates a HITS record for each of the seven COLUMN CONCEPTS, a HITS record for each of the twelve ROW CONCEPTS, and (seven times twelve), or eighty-four, HITS records for each of the PAIR CONCEPTs(i, j), i=1 to 7 and j=1 to 12. The total number of searched query terms and expressions of the same, and the number of HITS records showing the search result, is therefore, in the specific example shown in FIG. **2**, one hundred three. To have this coverage with the method of the prior art the user would have to manually perform one hundred one searches. The user would then have to write down or otherwise store, the results of each search. Even if the user managed to perform such a task, which would be a formidable job to complete, he or she would have expended considerable time that could have been otherwise used more effectively, and would not have the benefit of a single matrix formatted display of all the search results as is presented in FIG. **2**.

The above-identified step **104** searches for each ROW CONCEPT, each COLUMN CONCEPT and each PAIR CONCEPT(i, j) may be performed sequentially, or in parallel (i.e., concurrent), or as a combination of sequential and parallel. As known to persons skilled in the art of query searches, the selection between sequential and parallel depends in part on the structure of the database and the specific software design of the search engine. The current commercialized reductions to practice of the this invention use a sequential search, where COLUMN CONCEPT (**1**) is searched, a record of HITS(COLUMN CONCEPT (**1**)) is generated, and then COLUMN CONCEPT **2** is searched and so on until all of the COLUMN CONCEPTS have been searched. Related to the sequential search is a STOP SEARCH user interface button (not shown) appearing on the graphical user interface such as the example of FIG. **2**. As described above, in a preferred embodiment of the step **104** sequential search the HITS record is generated and the numbers K(CONCEPT) are displayed in sequence as the search progresses through the CONCEPTS. It is contemplated that particular choices of CONCEPTS and/or user-entered definitions of CONCEPTS, the definition process being described below, will yield HITS records necessitating a change of CONCEPTS. The STOP SEARCH button enables the user to stop the step **104** search accordingly.

Referring to FIGS. **2** and **3**, the process of defining CONCEPTS will be described. FIG. **2**, as described above, is an example graphical user interface **10** for performing the search step **104**. FIG. **3** shows an example graphical user interface **320** for defining the CONCEPTS. Field **320** includes a CONCEPT LIST **322** which lists all CONCEPTS that one or more users have defined. The list of CONCEPTS can be global, or can be particular to a user, the latter being written according to standard software design practice for multi-user programs having one or more parameter lists specific to particular users.

Field **322** of FIG. **3** displays the following CONCEPTS: "AIDS", "CHEMWPNS", "DRUGS", "HEARATK", "MDR", "MEXICAN", "OC", "TERRORISM", "USEAST", "USGSI", "USSOUTH", "USWEST", "VIOLENCE" and "WEATHER". The FIG. **3** example provides the user means to highlight one of the CONCEPTS in the list **322**, such as the "WEATHER" that is highlighted at **322**A in the figure. The code for displaying a list of entries in a file, scrolling through the displayed entries, and highlighting one or more of the list using, for example, the left click of a conventional computer mouse is well known in the relevant arts and, therefore, description is omitted. Field **324** displays the selected CONCEPT and field **326**, labeled EXPANSION, shows the Boolean expression that defines it.

Field **328** of FIG. **3** presents the user with a list of CONNECTIONS and EXPANDORS, which are Boolean operators and search query expandors/modifiers for connecting and modifying terms within the expression appearing in field **324**. Example CONNECTIONS in the FIG. **3** field **328** are: "AND", "OR" and "WITHIN#". Example EXPANDORS are: "Concept@", "Fuzzy %", "NOT", "PHONIC", "STEMMING", and "SYNONYMS". Each of the example CONNECTIONS is a Boolean logic term that is known in the art relating to this invention. The example EXPANDORS of "NOT", "PHONIC", "STEMMING" and "SYNONYMS" are likewise well known, and further description is therefore omitted. The example CONNECTION labeled "Concept@" denotes the CONCEPT following it as, itself, another defined CONCEPT. For example, a CONCEPT typed into field **324** as "ROAD" would be defined as the four letter string "R,O,A,D". The search step **104** would therefore look for this four letter string when searching the GDS databases. On the other hand, if the user had defined the CONCEPT "ROAD" using the FIG. **3** example graphical user interface, and then stored it in the list of CONCEPTS in field **322**, then subsequent use of that definition could be invoked by typing it as "@ROAD". An example definition of "ROAD" would be "street OR avenue OR boulevard". The example CONNECTION labeled "Fuzzy %", where "fuzzy" is defined according to its understood meaning in the arts pertaining to this invention.

As can be understood by persons skilled in the arts relating to this invention, the use, power and computational overhead relating to particular CONNECTIONS and EXPANDORS depends, in part, on the particular structure of the GDS databases selected at step **100**, in addition to the indexing scheme, if any.

Referring to the FIG. **3** example, the selected CONCEPT is "WEATHER", which is defined in field **66** as "storm or rain or winds or tornado or "hurricane or snow or "flood or storm surf or weather-related or NOAA or thunderstorms or El Nino". The tilda sign "~" is a wild card. The Boolean OR operator in field **328** connects the terms and, therefore, the expression in field **326** defines the CONCEPT "WEATHER" as any word or information that meets any term in the expression.

The CONCEPT definition field **320** of FIG. **3** allows a user to define existing words in a manner suited for the particular search, and to coin new CONCEPTS. For example, if a user wished to define "ski" to focus on water skiing and not include snow skiing it could be entered in the list **322** and defined in field **326** as SKI AND WATER AND BOAT. Alternatively, the user could coin a word, such as SNSKI, to have the same meaning.

FIG. **2** shows the search results of step **102** as a two-dimensional array of numbers, as ROW CELLs in field **22**, COLUMN CELLs in field **24**, and PAIR CELLS in field **28**, reflecting the HITS record obtained for each of the ROW CONCEPTS, COLUMN CONCEPTS, and PAIR CONCEPTS (i, j). Referring to FIG. **4**, an optional feature of the invention displays the K numbers for each HITS record as a three dimensional graph **400**, with the height of each cell graph, a representative one being labeled as **402**, representing the search results of the cell's associated COLUMN CONCEPT, ROW CONCEPT or PAIR CONCEPT(i, j). The FIG. **4** example sets the height of each cell graph **402** according to the number of documents or files meeting the criteria of the cell's corresponding CONCEPTS. The FIG. **4** display also includes a legend **404** which shows color codes of the CONCEPTS embodied by the COLUMN CONCEPTS and ROW CONCEPTS. The cell graphs **402** of each PAIR CONCEPT(i, j) may also be color coded as a blend of its constituent i^{th} ROW CONCEPT and j^{th} COLUMN CONCEPT.

Referring to the high level flow chart of FIG. **1**, and the graphical user interface FIGS. **2** and **5**, an analysis step **106** that is closely integrated with the search step **104** will be described. Step **106** can be used after each iteration of search step **104** and, based on commands received from the user, assists in analyzing the search using information provided by the HITS records generated at step **104**. It will be understood that describing step **106** as the "next step" is for purposes of describing an example operation of this invention. As will be understood, the analysis step **106**, as well as the later described steps of collection and reporting, do not have to be completed before re-running step **104**, either with different ROW CONCEPTS and COLUMN CONCEPTS, or after returning to steps **100** and **102** to select different GDS databases or indexing operations on the same. Also, as will be described, the present method permits the user to perform a "drill down" search on the documents within the HITS records in any of the ROW CELLS, COLUMN CELLS, or PAIR CELLS of FIG. **2**.

Referring to FIG. **2**, after the step **104** search the user is presented with a two dimensional array of HITS records, with the number of documents within the HITS record appearing in the cell corresponding in position to the CONCEPTS searched. As described above, each HITS record contains an identifier of each of the documents obtained by the search of the CONCEPT or pair of CONCEPTS corresponding to the cell. The user can then highlight any of the cells using, for example, the left click of a conventional computer mouse whereupon step **106** presents, in field **30** of FIG. **2**, the ordered list of documents identified in the record HITS corresponding to the selected cell. For example, in FIG. **2** PAIR CELL(**1**, **1**) in field **28** is highlighted. PAIR CELL(**1**, **1**) corresponds to PAIR CONCEPT(**1**, **1**), which is (heart, treatment). As shown by the number "14" appearing in PAIR CELL(**1**, **1**), seven documents were found in the "Medical" database index having information within the meaning of the "heart" CONCEPT and the "treatment" CONCEPT.

When the user highlights PAIR CELL(**1**, **1**) the display field **30** presents, in the FIG. **2** example, the first five documents in the HITS(heart, treatment) record. The display field **30** has, for each document listed, a field **30**A for the document order number, a field **30**B for the document's file name, a field **30**C for the size in bytes of the document, a field **30**D for the title of the document, if any exists, a field **30**E identifying a date associated with the document, and a field **30**F identifying the GDS database or database index name in which the document was found. As will be understood by one of ordinary skill, depending on the GDS

database searched, and whether or not the GDS was indexed, the documents may not have "names" and, in such cases, there may be no data appearing in field **30**B. Similarly, in some instances, there may be no date data appearing in the field **30**F.

The ordering of the documents listed in field **30** is a design choice, and may be in accordance with, for example, the "relevance" ordering used by Yahoo. In the particular example shown in FIG. **2**, the ordering is according to the number of occurrences, within the documents, of the search CONCEPTS for which the documents were obtained. Referring to FIG. **2**, the uppermost document, having a label of "1" in field **30**A, as it has the highest number of occurrences of the PAIR CONCEPT (heart, treatment) of any of the documents listed, namely "30" as appears in field **30**B.

Also in step **106** the user can click a MATRIX SPEED READING button (not shown) on the graphical user interface of FIG. **2** and will be presented with a new graphical user interface such as, for example MATRIX SPEED READING display **500** of FIG. **5**, allowing a quick reading of the documents within the HITS record for any cell in field **30** of FIG. **2**. As shown in the example MATRIX SPEED READING display **500** includes a field **502** and a field **504** listing, respectively, all of the COLUMN CONCEPTS and ROW CONCEPTS used in the search step **104**. Through a standard interface device such as, for example, a mouse (not shown) the user highlights one of the ROW CONCEPTS in field **502** and, if the user wishes to view the search results for a PAIR CONCEPT(i, j), also highlights one of the COLUMN CONCEPTS in field **504**. In the particular example shown in FIG. **5** the user has highlighted the ROW CONCEPT "heart" and the COLUMN CONCEPT "treatment", which presents a scroll list in field **30**, as described in reference to FIG. **2**, having short descriptions of the documents in the record HITS("heart", "treatment"). Field **506** scrolls through a more detailed description of the documents listed in field **30**.

Referring to FIG. **6**, another search example using the search matrix graphical user interface **10** of FIG. **2**, with a further result browser feature **600** of the invention, will be described. Field **16** of FIG. **6** lists ten particular ROW CONCEPTS including "@HOLOMEM", which is ROW CONCEPT(1), and "holograph", which is ROW CONCEPT **5**, and "HDSS", which is ROW CONCEPT(**10**). As described in reference to field **328** of the CONCEPT definition interface area **320** in FIG. **3**, the ampersand "@" prefix to "HOLOME" in ROW CONCEPT(**1**) means that the CONCEPT "HOLOME" is in the CONCEPT list of field **322** and, accordingly, it is defined according to a Boolean expression appearing in field **326**.

Referring to FIG. **6**, CONCEPT(**5**) is "holograph, which is a self-defined search query term of "holograph" including, as indicated by the tilda "~", a tail of any letter string.

A shown in FIG. **6**, the user has highlighted PAIR CELL(**1**, **2**), corresponding, in the example, to PAIR CONCEPT ("@HOLOMEM", "@HOLOGRAM"). The PAIR CELL(**1**, **2**) displays "45", meaning that the number of documents in this HITS record is forty-five. As described in reference to FIG. **2**, the user's highlighting of PAIR CELL(**1**, **2**) causes a list of the documents identified in HITS ("@HOLOME", "@HOLOGRAM") to appear as a scroll list in field **30**. The user can then highlight any of the documents in field **30**, whereupon a browser (not shown) retrieves the documents and displays it in field **600**. Field **600** may, depending on the design choice for the browser, be an overlay window appearing with the field **30** display.

As seen from the example text appearing in field **600** of FIG. **6**, the browser automatically highlights all occurrences of the search CONCEPTS within the selected document. The specific code-level design of the browser is readily generated by one of ordinary skill in the arts relating to this invention and, accordingly, description is omitted.

Referring to FIGS. **1**, **2** and **7**, a collection step **108** and organizing step **110** for systematically extracting information from user-selected documents within a HITS record, and placing that information into a convenient, consistent, user-specified template for ready insertion into a template report will be described.

First, as described referring to FIG. **2**, the user analyzes the search results by viewing the ROW CELLS, COLUMN CELLS and PAIR CELLS in fields **22**, **24** and **28**, clicking on those of interest, scrolling through the documents and reading the contents. The user can now generate a NOTES document, such as the example in FIG. **7**, which is a template onto which the user pastes information from the particular document which the user has selected and is reading using, for example, the browser feature described in reference to FIG. **5**. The user generates a NOTES document such as shown in FIG. **7** by clicking on the "NOTES" screen button **32** in FIG. **2**, which generates a pull-down from which the user can generate the document.

Referring to FIG. **7**, each NOTES document **700** is a summary profile of a particular document obtained by the search step **104**, the NOTES document **700** being formatted for the user to enter certain information in specific fields. An example formatting is shown in FIG. **7**, and consists of a HEADER **702**, a DOCUMENT TEXT field **704**, a DOCUMENT IMAGE field **706**, and a COMMENT field **708**. The HEADER field **702** includes TOPIC field **710**, a document DATE field **712**, a USER field **714**, a CATEGORY field **716**, a LABEL field **718**, a SOURCE field **720**, and an ATTACHED field **722**. Some of the HEADER fields **710**–**722** are automatically filled in, using the information in the HITs record that listed the subject document. These include the DATE field **712** and the SOURCE field **720**. The user selects and enters information into the TOPIC FIELD **710**, the CATEGORY field **716** and the LABEL field **718**, preferably using an outline hierarchy for the REPORT step **112** of FIG. **1**, described below, to assemble a plurality of NOTES into a REPORT.

Referring to FIGS. **6** and **7**, information from a document listed in FIG. **6** field **30**, and selected and displayed through the browser shown by FIG. **6** field **600** is moved into the NOTES document of FIG. **7** by a "drag and drop" process that, itself, is well-known in the relevant arts. More particularly, the user would select, typically by highlighting, text portions from the document appearing in the browser display **600**, and "drag and drop" those portions into the DOCUMENT TEXT field **714** of the NOTES document. If the document appearing in the browser display **600** is in HTML format, or another format supporting hyperlinks, the text appearing in field **714** will have those hyperlinks. If there are any images using, for example, .jpg or .tiff format, within the document appearing in the browser display **600** the user can insert these into the DOCUMENT IMAGE field **718**. After or during the "drag and drop" operations the user can type his or her own comments into the COMMENT field **708**.

In a typical search session using the method of the present invention the user may generate a plurality of, for example, ten FIG. **7** NOTES documents **700**.

As will be understood, the described method covers the entire knowledge management process of searching,

collecting, analyzing, organizing, and reporting. it provides the ability to conduct text searches across files, documents, Web pages, and databases located anywhere on the user's personal computer, network, or Internet. The present invention is not just a search toll; it includes the described set of analytical and reporting tools enabling the user to evaluate the results of searches, organize and create reports. The described method for searching a relational database(s) or a collection of unrelated documents and text files.

Search step **104** can be repeated, using the HITS records identified by any particular COLUMN CELL, ROW CELL or PAIR CELL as the universe of documents searched. The documents can be searched, by applying another set of COLUMN CONCEPTS and ROW CONCEPTS to the results of the first search. This can be repeated as often as the user wishes. This is commonly referred to as "drilldown".

The matrix search parameters, i.e., the GDS database, the ROW CONCEPTS and COLUMN CONCEPTS, and the HITs records of the results can be saved, using file storage methods well-known to persons of ordinary skill in the arts relating to this invention, for later use and shared with other users.

The matrix search parameters, i.e., the GDS database, the ROW CONCEPTS and COLUMN CONCEPTS, and the HITs records of the results can be exported/copy filed into other directories by selecting the result ROW CELLS, COLUMN CELLS and PAIR CELLS. This enables the user to reorganize his data into other directories and also send collections of data to other users.

The described Collecting step **108** enables the user to store full or partial text (selected by highlighting words) into digital filing system collections database, and to save multimedia type documents (e.g., .jpeg, .gif, .wav). The described DOCUMENTS TEXT field **704**, DOCUMENT IMAGE field **706** and COMMENT field **708** of the FIG. **7** NOTES documents **700** enables the user to include other related data into collection fields, store hyperlink back to the original source. In addition, the format of NOTES documents **700** allows them to be collected as a library (not shown) for rapid searching, using the HEADER **700** as an index. Further, the formatting of the NOTES documents allows step **112** to generate web-ready REPORTS from the collection database. In addition, a Create Link Analysis function of step **112** provides charts/reports that are linked back to items in the collection database. These Links reports can be shared with other users.

Referring to display **900** of FIG. **9**, a further embodiment of this invention uses the matrix of ROW CONCEPTS and COLUMN CONCEPTS entered, for example, through the FIG. **2** example interface, with a third party search engine. The FIG. **9** example uses the Northern Lights® search engine to search the Internet using the CONCEPTS of the FIG. **6** example. The CONCEPT matrix search process is in accordance with the FIG. **6** example, namely searching M ROW CONCEPTS(i), i=1 to M, searching N COLUMN CONCEPTS(j), j=1 to N, and then all M×N PAIR CONCEPTS(i, j), i=1 to M, j=1 to N. An additional operation is that the CONCEPTS are converted in to a form compatible with, for the FIG. **9** example, the Northern Lights search engine. More particularly, the field **901** shows that the specific PAIR CONCEPT searched in the FIG. **9** example is the PAIR CONCEPT ("@HOLOMEM", @HOLOGRP"). As described, the preferred logical operation for forming a PAIR CONCEPT from a ROW CONCEPT and a COLUMN CONCEPT is the AND operation. The PAIR CONCEPT ("@HOLOMEM", @HOLOGRP") is therefore, for this example, the CONCEPT "@HOLOMEM" ANDed with the CONCEPT "@HOLOGRP". To carry out the search, the method identifies the leading ampersand of each of the CONCEPTS and, in response, automatically expands each into its Boolean expression, as described in reference to FIG. **3**. The expression is then re-formatted for loading into field **902** which, for this example, accommodates the Northern Lights® search engine.

The specific example expansion is as follows: It is assumed that the CONCEPT of "@HOLOMEM" was previously defined using, for example, the graphical user interface of FIG. **3**, as ("holographicmemory" OR "holographic itprage" OR "holographicdata" OR "holographic recording" OR "HDSS" or "HolographicData Storage System" OR "optical storage" OR "spectral recording" OR "spherical memory"). The tilda "*" after "holographic" is, as described above, a wild card ending which encompasses for example, "holographical". It will also be assumed that "@HOLOGRP" was previously defined as ("IBM" OR "University of Dayton" OR "Lucent" OR "Bayer Corp" OR "Rockwell" OR "Kodak" OR "Stanford University"). The method therefore performs the search shown in FIG. **9** by ANDing the respective expansions of "@HOLOMEM" and "@HOLOGRP" and formatting the resulting expression as required by the Northern Light® search engine, as shown in field **902**. The formatting operation is readily implemented in conventional scripting code by one of ordinary skill in the arts relevant to this invention.

Referring to FIGS. **6** and **9**, the entire matrix of ROW CONCEPTS, COLUMN CONCEPTS and PAIR CONCEPTS shown in FIG. **6** is searched by formatting each CONCEPT, or PAIR CONCEPT such as, for example, the individual and PAIR CONCEPTS "@HOLOMEM" and @HOLOGRP", performing the search for each, and displaying the results in the matrix format of, for example, FIG. **2**. The search results are listed in field **904**. Each entry (not labeled) in field **904** has a hyperlink (not labeled) such as, for example, "HOLOGRAPHIC DATA STORAGE", which is entry "2", in accordance with the standard format of Northern Lights and similar pay-per-document search services. As known to one of ordinary skill in the arts pertaining to this invention, clicking on any of the hyperlinks downloads either a complete document or, as typical with services such as Northern Lights®, a summary, as shown in FIG. **10**. Referring to FIG. **10**, the user typically purchases the document by clicking on, for example, field **1000**. The resulting purchase operation is well known in the art.

It will be understood that, typically, the quick-reading feature described in reference to FIG. **5** will not be available when the matrix search of CONCEPTs interfaces to and uses pay-per-view search engines. Similarly, the browser feature of FIG. **6** is not available. Referring to FIGS. **6**, **7**, **11** and **12**, however, the previously described NOTES document generation can be used with typically free summaries provided by the third-party search services. More particularly, as shown in FIGS. **9**, **10** and **11**, pay-per-document search services such as, for example, Northern Lights typically download a summary of a document that a user clicks on, providing enough information to the user to allow a proper decision to purchase. FIG. **11** shows such a summary, which the service such as Northern Lights downloaded to the user in response to the user clicking on one of the hyperlinks in FIG. **9** field **904**, as described above. Fields **1100** and **1102** show the title and content of the summary, respectively, after being highlighted by the user. Referring to FIG. **12**, the content of the highlighted fields **1100** and **1102** from FIG. **11**

are inserted into fields **1202** and **1203**, respectively, of the generated NOTES document in the manner described in reference to FIG. **7**. Field **1204** of FIG. **12** contains the URL identifier of the summarized document. The user or another person can then use the NOTES document shown in FIG. **12** to retrieve and pay for a complete content of the document.

While the present invention has been disclosed with reference to certain preferred embodiments, these should not be considered to limit the present invention. One skilled in the art will readily recognize that variations of these embodiments are possible, each falling within the scope of the invention, as set forth in the claims below.

What is claimed is:

1. A method for searching electronic text files, comprising steps of:

providing a plurality of electronic text files;

providing a human interface apparatus for a user to access the plurality of electronic text files;

receiving, at the human interface apparatus, a plurality of M first search terms;

receiving, at the human interface apparatus, a plurality of N second search terms;

receiving a start search command data and, in response, performing the following steps:

(i) performing a key search of the plurality of electronic files and generating a first search term hit list associated with each of the M first search term, the first search term hit list identifying, for each of the M first search terms, the electronic text files from among said plurality of electronic text files having said first search term,

(ii) performing a key search of the plurality of electronic files and generating a second search term hit list associated with each of the N second search terms, the second search term hit list identifying, for each of the second search terms, the electronic text files from among said plurality of electronic text files having said second search term, and

(iii) identifying MxN unique search term pairs, each search term pair representing one first search term from each of the plurality of M first search terms and one second search term from the plurality of N second search terms; and

(iv) generating a plurality of MxN conjunction hit lists, each associated with each key word pair, each conjunction hit list identifying the electronic files from among the plurality of electronic text files having both the first search term and the second search term of the key word pair, and identifying said electronic files' population count.

2. A method according to claim **1** further comprising:

receiving a user-defined search term definition data, said definition data including a search term name and a Boolean expression of language words associated with said search term name;

displaying said search term name; and

receiving a user-entered command selecting said displays search term and, in response, assigning the Boolean expression associated with the said search term as one of said first search terms.

3. A method according to claim **1** further comprising steps of

receiving a search result format command from the user;

receiving an outline command and, in response generating a visual display of a report outline, the report outline having information formatted for comparison with the information field of one or more of said collection documents;

receiving a report generation command from the user and, in response, generating a report document based on a plurality of said collection documents and formatted in accordance with the report outline.

4. A method for searching electronic files according to claim **1**, further comprising:

displaying at least a sub-plurality of said M first search terms along a first region of a display screen, said first region extending in a first direction along said display screen;

displaying at least a sub-plurality of said N second search terms along a second region of a display screen, said second region extending in a second direction along said display screen,

displaying, within each of a plurality of regions on said display screen, an indicia representing a corresponding one of said plurality of conjunction hit lists, each region aligned along said first direction with one of said at least a sub-plurality of said M first search terms and aligned along said second direction with one of said at least a sub-plurality of said N second search terms, and the indicia within said region representing the conjunction hit list associated with the said one of said M first search terms and said one of said N second search terms.

5. A method for searching electronic files according to claim **4**, further comprising:

receiving a user-entered command identifying one from among said regions;

displaying a representation of at least a sub-plurality of the population of files associated with the conjunction hit list represented by said indicia within said region.

6. A method for searching electronic files according to claim **5**, further comprising:

receiving a user-entered file-selection command identifying one from among said population of files associated with the conjunction hit list represented by said indicia within said region,

displaying a portion of said file, said portion showing the first search term and the second search term of the key word pair defined by said first search term and said second search term.

7. A method for searching electronic files according to claim **6**, further comprising:

receiving a user-entered collection document creation command and, in response, creating a collection file and displaying a content field for entering data into said collection file;

receiving a user-entered text select command and, in response, copying said portion of said file identified by said user-entered file selection command into said content field, and inserting a file selection hyperlink into said content field to be visibly associated with said file from which said portion was copied.

8. A search method comprising:

providing a plurality of electronic text files;

providing a human interface apparatus for receiving search terms and commands from a user, and for accessing and searching said plurality of files in accordance with said data and commands;

receiving, at the human interface apparatus, a first search term;

receiving, at the human interface apparatus, a second search term;

receiving, at the human interface apparatus, a third search term;

receiving, at the human interface apparatus, a fourth search term;

receiving, at the human interface apparatus, a search command;

searching, in response to the search command, the plurality of electronic files and generating a first list, a second list, a third list, a fourth list and a fifth list, said first list identifying each of said electronic files having both the first and third search terms, said second list identifying each of said electronic files having both the first and fourth search terms, a third list, said third list identifying each of said electronic files having both the second and third search terms, and a fourth list, said fourth list identifying each of said electronic files having both the second and fourth search terms;

displaying, on a video display, said first search term, said second search term, said third search term and said fourth search term, a first list identifier representing at least a population count of electronic files within said first list, a second list identifier representing at least a population count of electronic files within said second list, a third list identifier representing at least a population count of electronic files within said third list, and a fourth list identifier representing at least a population count of electronic files within said fourth list.

9. A method according to claim 8, wherein

said displaying is performed such that said first list identifier is positioned and aligned according to said displayed first and third search terms, said second list identifier is positioned and aligned according to said displayed first and fourth search terms, said third list identifier is positioned and aligned according to said displayed second and third search terms, and said fourth list identifier is positioned and aligned according to said displayed second and fourth search terms.

* * * * *

US005819260A

United States Patent [19]

Lu et al.

[11] Patent Number: 5,819,260

[45] Date of Patent: *Oct. 6, 1998

[54] **PHRASE RECOGNITION METHOD AND APPARATUS**

[75] Inventors: **Xin Allan Lu**, Springboro; **David James Miller**, Dayton; **John Richard Wassum**, Springboro, all of Ohio

[73] Assignee: **Lexis-Nexis**, Miamisburg, Ohio

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **589,468**

[22] Filed: **Jan. 22, 1996**

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **707/3**; 707/1; 707/4; 707/5

[58] **Field of Search** 395/604, 603, 395/605, 600, 602; 707/3, 1, 4, 5

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,864,502	9/1989	Kucera et al.	704/8
4,868,750	9/1989	Kucera et al.	704/8
4,914,590	4/1990	Loatman et al.	704/8
4,931,936	6/1990	Kugiyama et al.	704/4
4,994,966	2/1991	Hutchins	704/9
5,123,103	6/1992	Ohtaki et al.	707/5
5,146,405	9/1992	Church	704/9
5,161,105	11/1992	Kugimiya et al.	704/4
5,225,981	7/1993	Yokogawa	704/2
5,251,316	10/1993	Anick et al.	707/101
5,265,065	11/1993	Turtle	707/4
5,287,278	2/1994	Rau	704/1
5,289,375	2/1994	Fukumochi et al.	704/2
5,297,042	3/1994	Morita	707/5
5,299,124	3/1994	Fukumochi et al.	704/2
5,410,475	4/1995	Lu et al.	704/1
5,418,948	5/1995	Turtle	707/4
5,481,742	1/1996	Worley et al.	395/200.76
5,488,725	1/1996	Turtle	707/5

OTHER PUBLICATIONS

Salton et al., "A Simple Syntactic Approach for the Generation of Indexing Phrases", Technical Report, Department of Computer Science, Cornell University, Ithaca, New York, pp. 1–8 (note this reference is cited on form PTO 1449), Jul. 1990.

Salton, Gerard, et al., "A Simple Syntactic Approach for the Generation of Indexing Phrases", Technical Report, Department of Computer Science, Cornell University, Ithaca, New York, Jul. 1990.

Coates–Stephens, Sam, "The Analysis and Acquisition of Proper Names for the Understanding of Free Text", *Computers and the Humanities,* Kluwer Academic Publishers, the Netherlands, vol. 26, pp. 441–456, 1993.

Evans, David A., et al., "Automatic Indexing Using Selective NLP and First Order Thesauri", *Intelligent Text and Image Handling, Proceedings of a Conference on Intelligent Text and Image Handling 'RIAO 91'*, Barcelona, Spain, Apr. 1991.

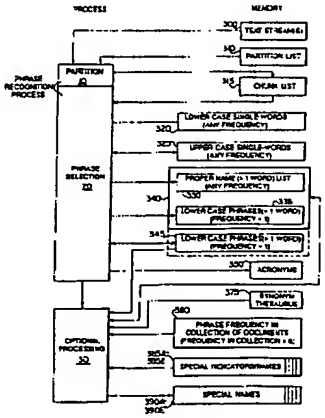
(List continued on next page.)

Primary Examiner—Thomas G. Black
Assistant Examiner—Greta L. Robinson
Attorney, Agent, or Firm—Reid & Priest LLP

[57] **ABSTRACT**

A phrase recognition method breaks streams of text into text "chunks" and selects certain chunks as "phrases" useful for automated full text searching. The phrase recognition method uses a carefully assembled list of partition elements to partition the text into the chunks, and selects phrases from the chunks according to a small number of frequency based definitions. The method can also incorporate additional processes such as categorization of proper names to enhance phrase recognition. The method selects phrases quickly and efficiently, referring simply to the phrases themselves and the frequency with which they are encountered, rather than relying on complex, time-consuming, resource-consuming grammatical analysis, or on collocation schemes of limited applicability, or on heuristical text analysis of limited reliability or utility.

30 Claims, 6 Drawing Sheets



OTHER PUBLICATIONS

Ahlswede, Thomas, et al., "Automatic Construction of a Phrasal Thesaurus for an Information Retrieval System from a Machine Readable Dictionary", *Proceedings of RIAO '88,* Cambridge, Massachusetts, Mar. 1988, pp. 597–608.

Chruch, Kenneth Ward, et al. (Bell Laboratories), "A Stochastic Parts Program and Noun Phrases Parser for Unrestricted Text", *Proceedings of 1989 International Conference on Acoustics, Speech and Signal Processing* (IEEE Cat. No. 89CH2673–2), Glasgow, Scotland, UK, May 1989, pp. 695–698.

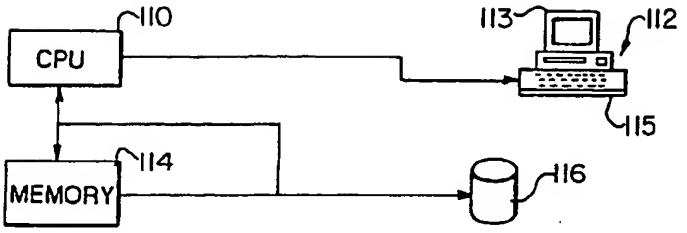

110
CPU
113
112
115
114
MEMORY
116

FIG. 1

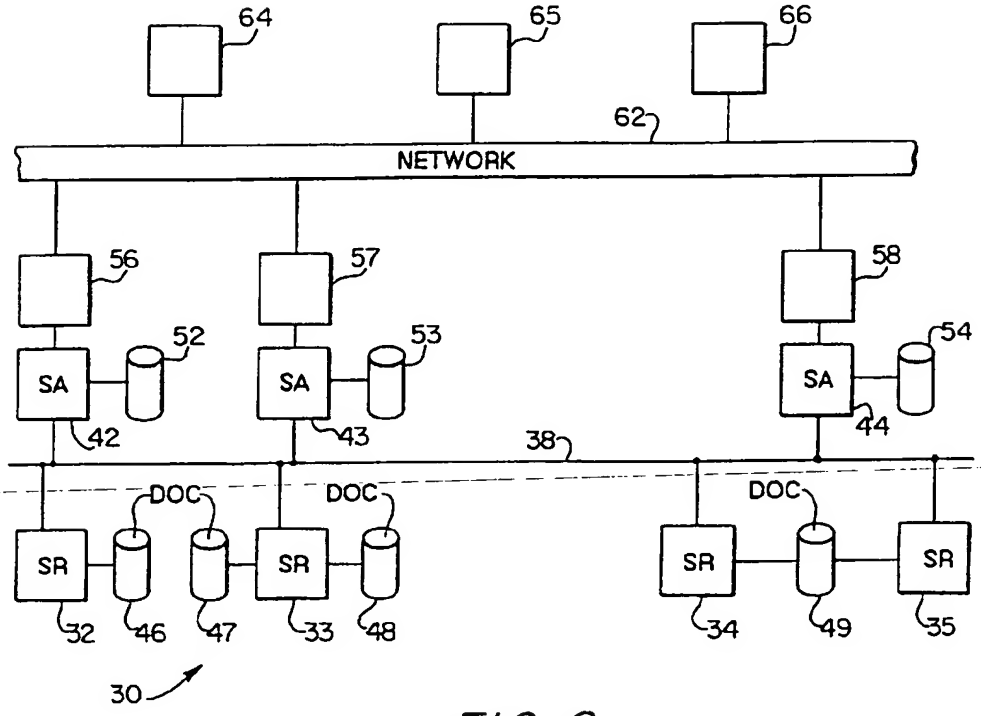

64
65
66
62
NETWORK
56
57
58
52
53
54
SA
SA
SA
42
43
44
38
DOC
DOC
DOC
SR
SR
SR
32
46
47
33
48
34
49
35
30

FIG. 2

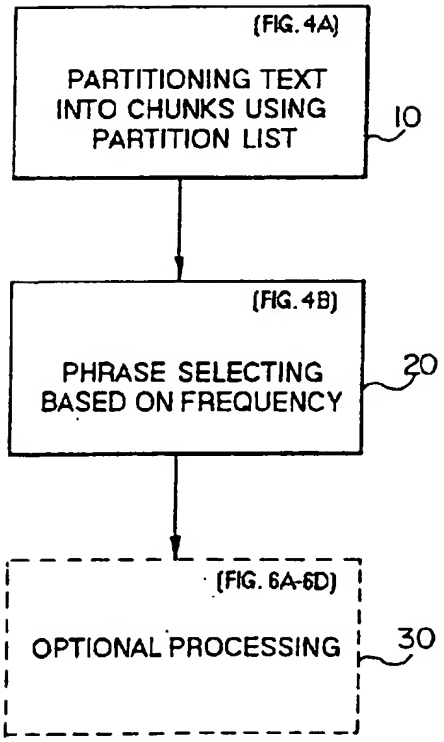

(FIG. 4A)
PARTITIONING TEXT INTO CHUNKS USING PARTITION LIST
10
(FIG. 4B)
PHRASE SELECTING BASED ON FREQUENCY
20
(FIG. 6A-6D)
OPTIONAL PROCESSING
30

FIG. 3

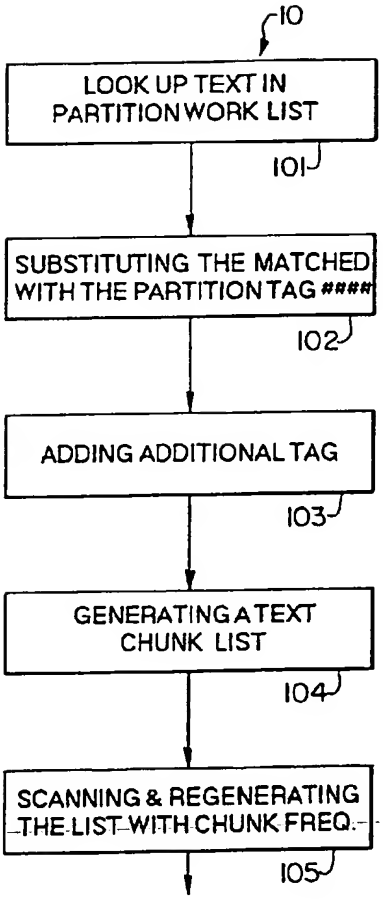

10
LOOK UP TEXT IN PARTITION WORK LIST
101
SUBSTITUTING THE MATCHED WITH THE PARTITION TAG ####
102
ADDING ADDITIONAL TAG
103
GENERATING A TEXT CHUNK LIST
104
SCANNING & REGENERATING THE LIST WITH CHUNK FREQ.
105

FIG. 4A

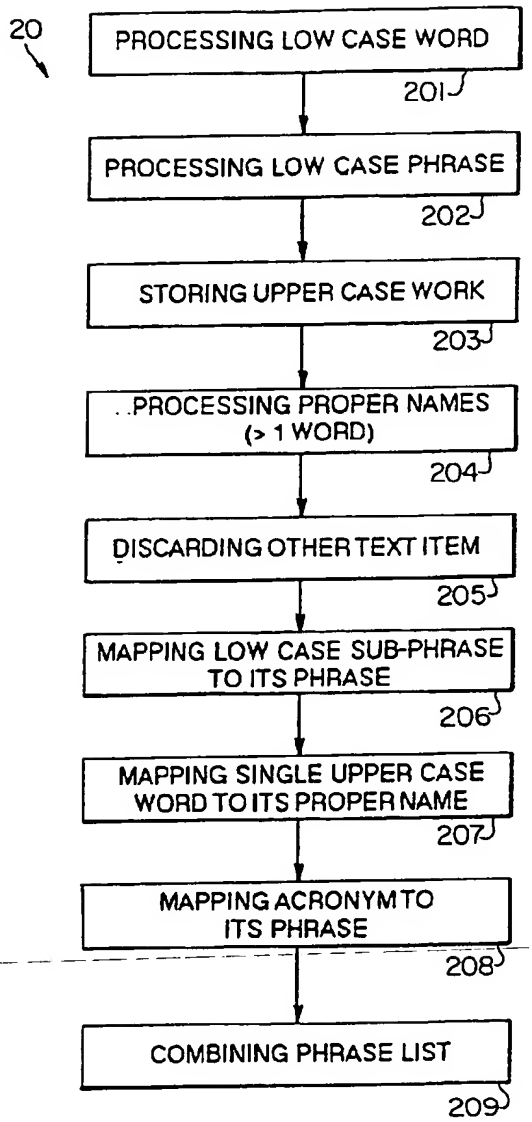

20
PROCESSING LOW CASE WORD
201
PROCESSING LOW CASE PHRASE
202
STORING UPPER CASE WORK
203
PROCESSING PROPER NAMES (> 1 WORD)
204
DISCARDING OTHER TEXT ITEM
205
MAPPING LOW CASE SUB-PHRASE TO ITS PHRASE
206
MAPPING SINGLE UPPER CASE WORD TO ITS PROPER NAME
207
MAPPING ACRONYM TO ITS PHRASE
208
COMBINING PHRASE LIST
209

FIG. 4B

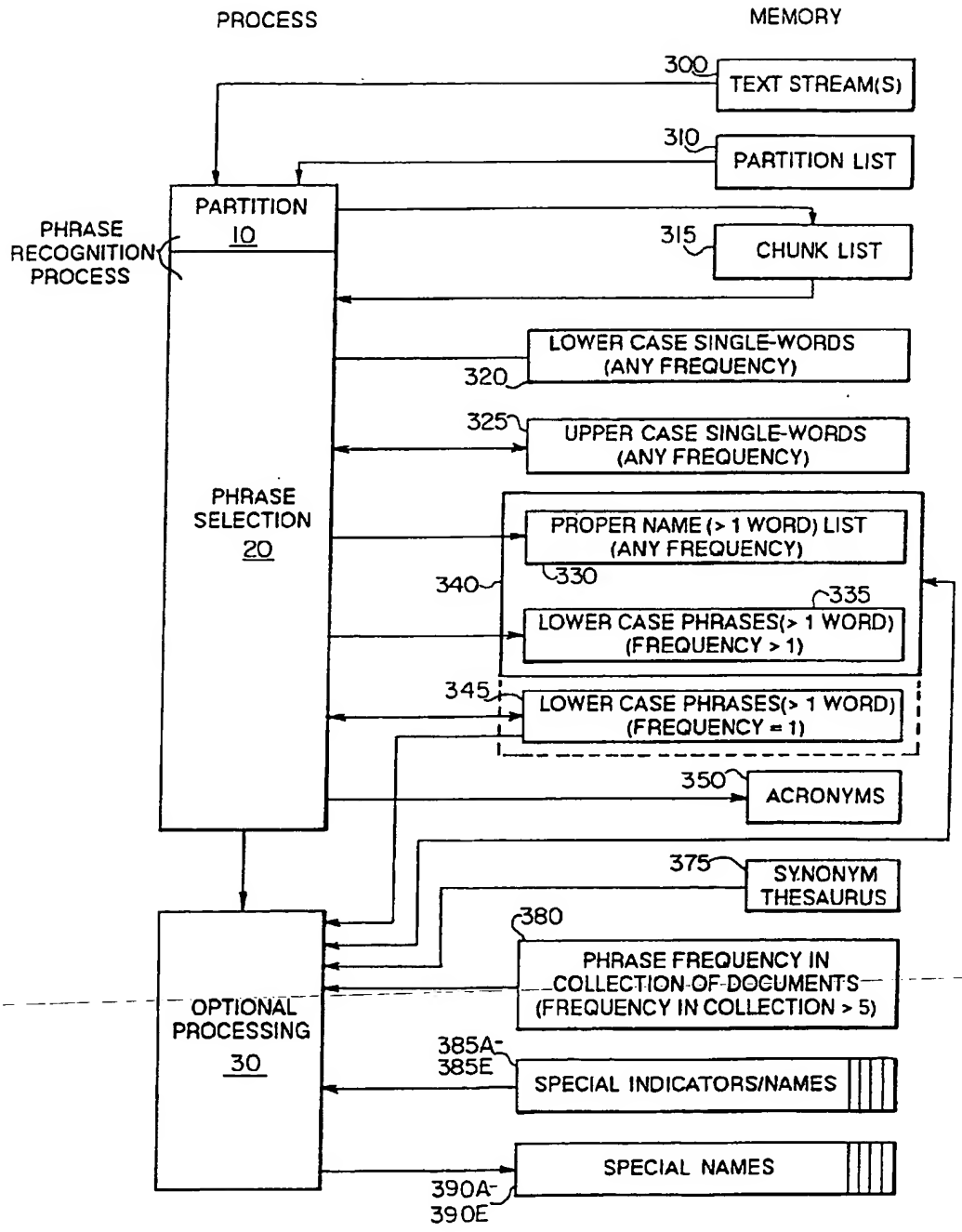

PROCESS
MEMORY
300
TEXT STREAM(S)
310
PARTITION LIST
PARTITION
10
PHRASE
RECOGNITION
PROCESS
315
CHUNK LIST
LOWER CASE SINGLE-WORDS
(ANY FREQUENCY)
320
325
UPPER CASE SINGLE-WORDS
(ANY FREQUENCY)
PHRASE
SELECTION
20
PROPER NAME (> 1 WORD) LIST
(ANY FREQUENCY)
330
340
335
LOWER CASE PHRASES(> 1 WORD)
(FREQUENCY > 1)
345
LOWER CASE PHRASES(> 1 WORD)
(FREQUENCY = 1)
350
ACRONYMS
375
SYNONYM
THESAURUS
380
PHRASE FREQUENCY IN
COLLECTION-OF-DOCUMENTS
(FREQUENCY IN COLLECTION > 5)
OPTIONAL
PROCESSING
30
385A-
385E
SPECIAL INDICATORS/NAMES
SPECIAL NAMES
390A-
390E

FIG. 5

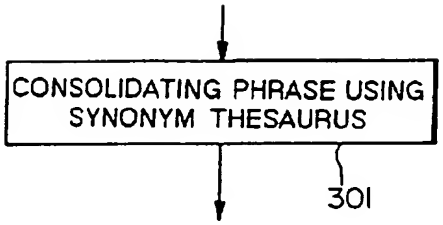


FIG. 6A

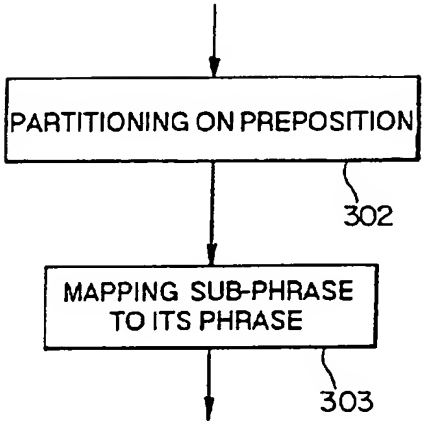


FIG. 6B

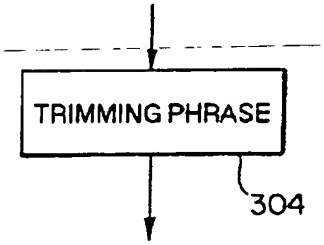


FIG. 6C

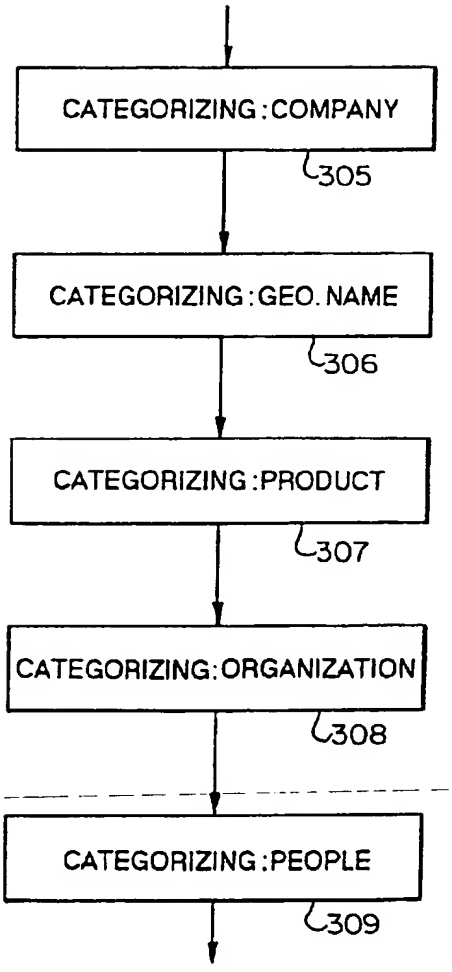


FIG. 6D

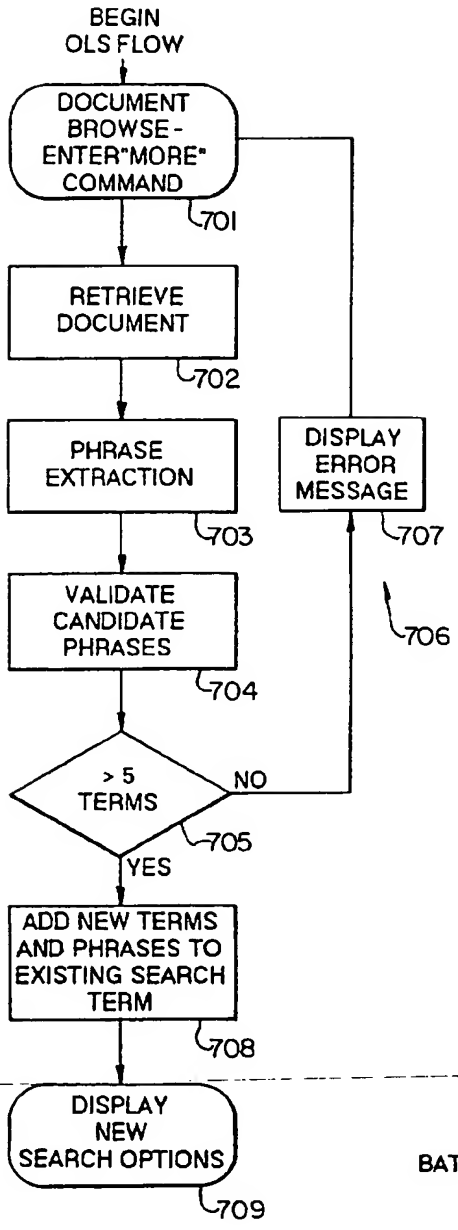


FIG. 7A

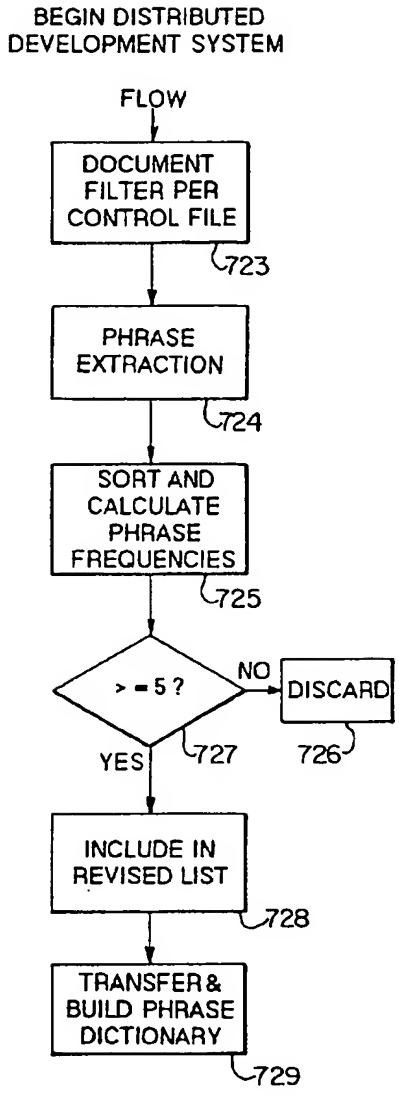


FIG. 7B

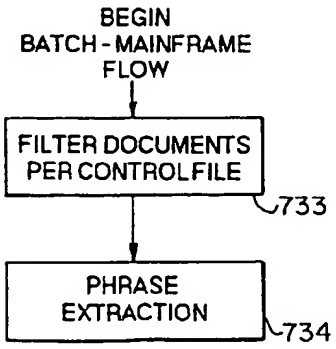


FIG. 7C

PHRASE RECOGNITION METHOD AND APPARATUS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to automated indexing of full-text documents to identify the content-bearing terms for later document retrieval. More specifically, the invention relates to computer-automated identification of phrases which are useful in representing the conceptual content of documents for indexing and retrieval.

2. Related Art

A type of content-bearing term is the "phrase", a language device used in information retrieval to improve retrieval precision. For example, the phrase "product liability" indicates a concept that neither of the two component words can fully express. Without this phrase, a retrieval process is unable to find the documents in which the concept is discussed.

In traditional Boolean retrieval systems, phrase recognition is not an issue. The systems are known as post-coordination indexing systems in that phrases can be discovered through examining the adjacency relationships among search words during the process of merging inverted lists associated with the words.

However, in modern information retrieval systems, the statistical distribution characteristics of index terms are crucial to the relevance ranking process, and it is desirable to recognize phrases and derive their statistical characteristics in advance. In addition, in fabricating hypertext databases, recognized phrases are necessary for hypertext linkage.

Known phrase recognition methods include three types: machine translation, statistical text analysis and heuristical text analysis.

First, machine translation's approach to recognizing phrases (known as compound structures) is to analyze part-of-speech tags associated with the words in an input text string, usually a sentence. Noun phrases and verb phrases are two examples of such phrases. Syntactical context and lexical relationships among the words are key factors that determine successful parsing of the text. In machine translation, the goal is not of finding correct phrases, but of discovering the correct syntactical structure of the input text string to support other translation tasks. It is infeasible to use this syntactical parsing method for processing commercial full-text databases; the method is inefficient and is in practical terms not scalable. Regarding machine translation, reference may be made to U.S. Pat. Nos. 5,299,124, 5,289,375, 4,994,966, 4,914,590, 4,931,936, and 4,864,502.

The second method of analysis, statistical text analysis, has two goals: disambiguating part of speech tagging, and discovering noun phrases or other compound terms. The statistics used include collocation information or mutual information, i.e., the probability that a given pair of part-of-speech tags or a given pair of words tends to appear together in a given data collection. When a word has more than one part-of-speech tag associated with it in a dictionary, consulting the part of speech of the next word and calculating the probability of occurrence of the two tags would help select a tag. Similarly, a pair of words that often appear together in the collection is probably a phrase. However, statistical text analysis requires knowledge of collocation that can only be derived from an known data collection. Disadvantageously, the method is not suitable for processing unknown data. Regarding statistical text analysis, reference may be made to U.S. Pat. Nos. 5,225,981, 5,146,405, and 4,868,750.

The third method of analysis, heuristical text analysis, emphasizes textual pattern recognition. Textual patterns include any recognizable text strings that represent concepts, such as company names, peoples' names, or product names. For example, a list of capital words followed by a company indicator like "Limited" or "Corp" is an example pattern for recognizing company names in text. The heuristical text analysis method requires strong observation ability from a human analyst. Due to the limitation of humans' observation span, heuristical text analysis is only feasible for small subject domains (e.g., company name, product names, case document names, addresses, etc.). Regarding heuristical text analysis, reference may be made to U.S. Pat. Nos. 5,410,475, 5,287,278, 5,251,316, and 5,161,105.

Thus, machine translation methods, involving potentially complex grammatical analysis, are too expensive and too error-prone for phrase recognition. Statistical text analysis, being based on collocation and being purely based on statistics, is still expensive because of the required full scale of part-of-speech tagging and pre-calculating collocation information, and also has difficulties processing unknown data without the collocation knowledge. Finally, heuristical text analysis, relying on "signal terms", is highly domain-dependent and has trouble processing general texts.

Thus, there is a need in the art for a simple, time-efficient, resource-efficient, and reliable phrase recognition method for use in assisting text indexing or for forming a statistical thesaurus. It is desired that the phrase recognition method be applicable to both individual documents and to large collections of documents, so that the performance of not only real-time on-line systems, but also distributed and mainframe text search systems can be improved. It is also desired that the phrase recognition method have engineering scalability, and not be limited to particular domains of knowledge.

The present invention is directed to fulfilling these needs.

SUMMARY OF THE INVENTION

The present invention provides a phrase recognition method which breaks text into text "chunks" and selects certain chunks as "phrases" useful for automated full text searching. The invention uses a carefully assembled list of partition words to partition the text into the chunks, and selects phrases from the chunks according to a small number of frequency-based definitions. The invention can also incorporate additional processes such as categorization of proper names to enhance phrase recognition.

The invention achieves its goals quickly and efficiently, referring simply to the phrases and the frequency with which they are encountered, rather than relying on complex, time-consuming, resource-consuming grammatical analysis, or on collocation schemes of limited applicability, or on heuristical text analysis of limited reliability or utility.

Additional objects, features and advantages of the invention will become apparent when the following Detailed Description of the Preferred Embodiments is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is better understood by reading the following Detailed Description of the Preferred Embodiments with reference to the accompanying drawing figures, in which like reference numerals refer to like elements throughout, and in which:

FIG. 1 illustrates an exemplary hardware configuration on which the inventive phrase recognition method may be executed.

FIG. 2 illustrates another exemplary hardware environment in which the inventive phrase recognition method may be practiced.

FIG. 3 is a high level flow chart schematically indicating execution in an embodiment of the phrase recognition method according to the present invention.

FIG. 4A is a flow chart schematically indicating execution in a module for partitioning text and generating text chunks.

FIG. 4B is a flow chart indicating execution of a module for selecting phrases using the data memory structure diagram of FIG. 5.

FIG. 5 is a data memory structure diagram schematically illustrating data flow during the inventive phrase recognition method (FIGS. 3, 4A, 4B) and corresponding memory allocation for various types of data used in accordance with the process.

FIG. 6A is a flow chart of an optional processing module for consolidating with a thesaurus.

FIG. 6B is a flow chart of an optional processing module for processing phrases with prepositions.

FIG. 6C is a flow chart of an optional processing module for trimming phrases with their collection frequencies.

FIG. 6D is a flow chart of an optional processing module for categorizing proper names.

FIGS. 7A, 7B and 7C illustrate exemplary applications of the inventive phrase recognition method according to the present invention. In particular: FIG. 7A indicates a user's viewing of a document in accordance with a suitable on-line text search system, and invoking the inventive phrase recognition method to search for additional documents of similar conceptual content; FIG. 7B schematically illustrates implementation of the phrase recognition method in a batch phrase recognition system in a distributed development system; FIG. 7C schematically illustrates application of the inventive phrase recognition method in a batch process in a mainframe system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In describing preferred embodiments of the present invention illustrated in the drawings, specific terminology is employed for the sake of clarity. However, the invention is not intended to be limited to the specific terminology so selected, and it is to be understood that each specific element includes all technical equivalents which operate in a similar manner to accomplish a similar purpose.

The concept of the present invention is first described on a particular example of a text stream. Then, block diagrams and flow charts are described, which illustrate non-limiting embodiments of the invention's structure and function.

Very briefly, a preferred embodiment of the inventive method partitions an input text stream based on punctuation and vocabulary. As the method processes the text stream sequentially, it inserts partition symbols between words if certain punctuation exists, such as a comma, end of sentence, or change in capitalization. Further, each word encountered is checked against one or more vocabulary lists, and may be discarded and replaced by a partition symbols, based on the word and where it is encountered.

After the document is thus processed, a set of candidate terms and "phrases" (a series of non-partitioned words) is produced. In the preferred embodiment, solitary words (individual words immediately surrounded by partitions) are ignored at this point. The phrases are processed to determine which phrases occur with higher frequency. Preferably, shorter phrases which occur with higher frequency and as subsets of lower-frequency but lengthier phrases are also sought. A set of phrases meeting or exceeding a given threshold frequency is produced.

The inventive method is more easily understood with reference to a particular example.

As mentioned above, members of a list of words (including punctuation) serve as "break points" to form text "chunks" within input text. A first (rudimentary) list includes words can be used as "stop words". The stop words usually carry little semantic information because they exist merely for various language functions. This list has a few hundred members and includes articles (e.g., "a", "the"), conjunctions (e.g., "and", "or"), adverbs (e.g., "where", "why"), prepositions (e.g., "of", "to", "for"), pronouns (e.g., "we", "his"), and perhaps some numeric items.

However, this first list is too short for the present phrase recognition method because it causes generation of a list of text chunks that are too long to allow efficient generation of desirable phrases. Additional stop words or other partition items are needed for reducing the size of the text chunks, so that more desirable phrases may be found.

The following example of text illustrates this problem. In this example, the text "chunks" are within square brackets, with the text chunks being separated by an members of the list of stop words (break points):

> [Citing] what is [called newly conciliatory comments] by the [leader] of the [Irish Republican Army]'s [political wing], the [Clinton Administration announced today] that it would [issue] him a [visa] to [attend] a [conference] on [Northern Ireland] in [Manhattan] on [Tuesday]. The [Administration] had been [leaning] against [issuing] the [visa] to the [official], [Gerry Adams], the [head] of [Sinn Fein], [leaving] the [White House caught] between the [British Government] and a [powerful bloc] of [Irish-American legislators] who [favored] the [visa]. (Parsed text based on rudimentary list)

Since desirable phrases include noun phrases (e.g., "ice cream"), adjective-noun phrases (e.g., "high school"), participle-noun phrases (e.g., "operating system"), and proper names (e.g., "White House"), it is safe to add adverbs (e.g., "fully") and non-participle verbs (e.g., "have", "is", "obtain") to the list of stop words to form an enhanced stop word list. This enhanced stop word list allows the method to provide smaller text chunks, yet is still compact enough for efficient look-up by computer. With the enhanced list, the above example text is parsed into chunks and stop words as follows:

> [Citing] what is [called] newly [conciliatory comments] by the [leader] of the [Irish Republican Army]'s

[political wing], the [Clinton Administration] announced [today] that it would issue him a [visa] to attend a [conference] on [Northern Ireland] in [Manhattan] on [Tuesday]. The [Administration] had been [leaning] against [issuing] the [visa] to the [official], [Gerry Adams], the [head] of [Sinn Fein], [leaving] the [White House] caught between the [British Government] and a [powerful bloc] of [Irish-American legislators] who favored the [visa]. (Second parsed text based on enhanced list)

The theoretical justification of using this enhanced list derives from two sources.

A first justification is that this list only represents about 13% of unique words in a general English dictionary. For example, in the Moby dictionary of 214,100 entries, there are 28,408 words that can be put into the list. This fact ensures that semantic, information in texts is maintained at a maximum level.

A second justification involves the lexical characteristics of these words. Most of the words bear little content. This second fact reduces the risk of losing semantic information in the text.

The basic concept of the invention having been described, particular implementations of its structure and function are now presented.

As will readily be appreciated, the invention is preferably embodied as software, instruction codes capable of being executed by digital computers, including commercially available general purpose digital computers well known to those skilled in the art. The particular hardware on which the invention may be implemented varies with the particular desired application of the inventive phrase recognition method. Three examples of the such application of the phrase recognition method are described in greater detail below, with reference to FIGS. 7A, 7B, and 7C. Briefly, the dynamic recognition method involved in an on-line system (FIG. 7A) may be implemented in IBM 370 assembly language code. Alternatively, in a batch recognition system in a distributed development system (FIG. 7B), the phrase recognition method may be implemented on a SUN work station using the PERL script interpretive prototyping language. As a still further implementation, the inventive phrase recognition method may be implemented on an Amdahl AMD 5995-1400-a mainframe so that another batch phrase recognition system (FIG. 7C) may be realized. Of course, the scope of the invention should not be limited by these exemplary embodiments or applications.

Embodiments of the inventive phrase recognition method may be implemented as a software program including a series of executable modules on a computer system. As shown in FIG. 1, an exemplary hardware platform includes a central processing unit **110**. The central processing unit interacts with a human user through a user interface **112**. The user interface is used for inputting information into the system and for interaction between the system and the human user. The user interface **112** includes, for example, a video display **113** and a keyboard **115**. A computer memory **114** provides storage for data and software programs which are executed by the central processing unit **110**. Auxiliary memory **116**, such as a hard disk drive or a tape drive, provides additional storage capacity and a means for retrieving large batches of information.

All components shown in FIG. **1** are of a type well known in the art. For example, the FIG. 1 system may include a SUN® work station including the execution platform Sparc 2 and SUN OS Version 4.1.2., available from SUN MICROSYSTEMS of Sunnyvale, Calif. Of course, the system of the present invention may be implemented on any number of modern computer systems.

A second, more complex environment in which the inventive phrase recognition method may be practiced is shown in FIG. **2**. In particular, a document search and retrieval system **30** is shown. The system allows a user to search a subset of a plurality of documents for particular key words or phrases. The system then allows the user to view documents that match the search request. The system **30** comprises a plurality of Search and Retrieval (SR) computers **32–35** connected via a high speed interconnection **38** to a plurality of Session Administrator (SA) computers **42–44**.

Each of the SR's **32–35** is connected to one or more document collections **46–49**, each containing text for a plurality of documents, indexes therefor, and other ancillary data. More than one SR can access a single document collection. Also, a single SR can be provided access to more than one document collection. The SR's **32–35** can be implemented using a variety of commercially available computers well known in the art, such as Model EX100 manufactured by Hitachi Data Systems of Santa Clara, Calif.

Each of the SA's **42–44** is provided access to data representing phrase and thesaurus dictionaries **52–54**. The SA's **42–44** can also be implemented using a variety of commercially available computers, such as Models 5990 and 5995 manufactured by Amdahl Corporation of Sunnyvale Calif. The interconnection **38** between the SR's and the SA's can be any one of a number of two-way high-speed computer data interconnections well known in the art, such as the Model 7200-DX manufactured by Network Systems Corporation of Minneapolis, Minn.

Each of the SA's **42–44** is connected to one of a plurality of front end processors **56–58**. The front end processors **56–58** provide a connection of the system **30** one or more commonly available networks **62** for accessing digital data, such as an X.25 network, long distance telephone lines, and/or SprintNet. Connected to the network **62** are plural terminals **64–66** which provide users access to the system **30**. Terminals **64–66** can be dumb terminals which simply process and display data inputs and outputs, or they can be one of a variety of readily available stand-alone computers, such as IBM or IBM-compatible personal computers. The front end processors **56–58** can be implemented by a variety of commercially available devices, such as Models 4745 and 4705 manufactured by the Amdahl Corporation of Sunnyvale Calif.

The number of components shown in FIG. **2** are for illustrative purposes only. The system **30** described herein can have any number of SA's, SR's, front end processors, etc. Also, the distribution of processing described herein may be modified and may in fact be performed on a single computer without departing from the spirit and scope of the invention.

A user wishing to access the system **30** via one of the terminals **64–66** will use the network **62** to establish a connection, by means well known in the art, to one of the front end processors **56–58**. The front end processors **56–58** handle communication with the user terminals **64–66** by providing output data for display by the terminals **64–66** and by processing terminal keyboard inputs entered by the user. The data output by the front end processors **56–58** includes text and screen commands. The front end processors **56–58** support screen control commands, such as the commonly known VT100 commands, which provide screen functionality to the terminals **64–66** such as clearing the screen and moving the cursor insertion point. The front end processors **56–58** can handle other known types of terminals and/or stand-alone computers by providing appropriate commands.

Each of the front end processors **56–58** communicates bidirectionally, by means well known in the art, with its corresponding one of the SA's **42–44**. It is also possible to configure the system, in a manner well known in the art, such that one or more of the front end processors can communicate with more than one of the SA's **42–44**. The front end processors **56–58** can be configured to "load balance" the SA's **42–44** in response to data flow patterns. The concept of load balancing is well known in the art.

Each of the SA's **42–44** contains an application program that processes search requests input by a user at one of the terminals **64–66** and passes the search request information onto one or more of the SR's **32–35** which perform the search and returns the results, including the text of the documents, to the SA's **42–44**. The SA's **42–44** provide the user with text documents corresponding to the search results via the terminals **64–66**. For a particular user session (i.e. a single user accessing the system via one of the terminals **64–66**), a single one of the SA's **42–44** will interact with a user through an appropriate one of the front end processors **56–58**.

Preferably, the inventive phrase recognition method is implemented in the session administrator SA computers **42–44**, with primary memory being in the SA computer itself and further memory being illustrated within elements **52–54**.

The principles on which the inventive method is based, and hardware systems and software platforms on which it may be executed, having been described, a preferred embodiment of the inventive phrase recognition method is described as follows.

FIG. **3** is a high level flow diagram of the phrase recognition method of the preferred embodiment.

Referring to FIG. **3**, the invention uses a carefully assembled list of English words (and other considerations such as punctuation) in a Partition Word List (more generally referred to as a Partition Entity List, or simply Partition List) to partition one or more input text streams into many text chunks. This partitioning process is illustrated in block **10**.

Block **20** indicates selection of phrases from among the chunks of text, according to frequency based definitions. A Phrase List, including the selected phrases, results from execution of the process in block **20**. During the phrase selection process, solitary words (single-word chunks), as well as words from the decomposed phrases, can be maintained separate from the Phrase List as optional outputs for other indexing activities.

Details of processes **10** and **20** are described with reference to FIGS. **4A** and **4B**.

The invention can optionally incorporate one or more other processes, generically indicated as element **30**. Such optional process may include categorization (examples described with reference to FIGS. **6A–6D**) to enhance the list of recognized phrases.

FIG. **4A** is a flow chart of FIG. **3** module **10**, for partitioning text and generating text chunks.

FIG. **4A** shows how the method, given a piece of text, partitions the text into many small text chunks. A critical component in this method is the Partition List (including words and punctuation) whose members serve as break points to generate the text chunks.

As mentioned above, a Partition List ideally allows parsing of text into short phrases, but is itself still compact enough for efficient computer look-up during the parsing process. Preferably, the Partition List is generated using not only articles, conjunctions, adverbs, prepositions, pronouns, and numeric items, but also adverbs and verbs, to form an enhanced list.

The text partitioning process starts off with looking up encountered text in the Partition List (at block **101**) and replacing every matched partition word or other partition entity with a partition tag such as "####" (shown at block **102**).

Additional partition tags are added into those text chunks at the point where there is a case change, either from lower case to upper case or vice versa (shown at block **103**). Block **104** indicates generation of the text chunk list which preserves the natural sequence of the chunks as encountered in the text.

The frequency information for each chunk in the list is collected by scanning the text chunks in their natural sequence. The first occurrence of each unique chunk in the sequence is registered as a new entry with its frequency as 1. Subsequent occurrences are registered by incrementing its frequency count by 1. This generation of occurrence frequencies in association with the respective chunks is indicated by block **105**.

FIG. **4B** is a flow chart of FIG. **3** module **20**, illustrating details of a preferred process for selecting which chunks are phrases.

FIG. **5** is a data memory structure diagram showing how data may be arranged in memory for the process, and how data flows into and out of various steps of the process. More specifically, the steps from FIG. **3** of text partitioning **10**, phrase selection **20**, and optional processing **30** (reproduced on the left side of FIG. **5**) are illustrated in conjunction with an exemplary data memory structure diagram (on the right side of FIG. **5**) to schematically illustrate data flow between major functional procedures and data structures. The various lists shown in the exemplary memory blocks on the right side of FIG. **5** are understood to include list members in conjunction with their respective frequencies of occurrence.

The memory (broadly, any data storage medium such as RAM and/or magnetic disk and/or optical disk and/or other suitable computer readable medium) may be structured in memory blocks as schematically illustrated. A text stream file **300** and a Partition List **310** are used as inputs to the partitioning process **10** of the inventive phrase recognition method. The partitioning process **10** provides a chunk list (understood as including corresponding chunk frequencies) **315**. Chunk list **315** is used by the phrase selection process **20** of the inventive phrase recognition method.

The partitioning process produces various groupings of chunks, each with their respective frequencies of occurrence within the text stream. These groupings of chunks are illustrated on the right side of FIG. **5**, with the understanding that the invention should not be limited to the particular memory structure so illustrated.

Specifically, lower case words (that is, single-word chunks) are in memory block **320**, capitalized or "allcaps" single-word chunks are in memory block **325**, a Proper Name List (preferably of greater than one word, each being capitalized or in allcaps) is in memory block **330**, lower case phrases of greater than one word occurring more than once are in memory block **335**, lower case phrases of greater than one word which were encountered only once are in memory block **345**, and, optionally, acronyms are in memory block **350**.

A synonym thesaurus in memory block **375** may be used in an optional process **30**. A phrase frequency list derived from a collection of plural documents in which the phrase frequency throughout the collection is greater than a threshold, in memory block **380**, may also be used in an optional processing procedure **30**. Further, one or more special indicator lists, generally indicated as **385A–385E**

(company indicators, geographic names, product names, organization indicators, English first names, respectively, some of which are exemplified in the attached List) may contribute to certain optional categorizing processes, and result in corresponding name lists (company names, geographic location names, product names, organization names, and English names) generally indicated as **390A–390E**.

Referring again to FIG. **4B**, after the text chunk list is produced, it is the time to make decision whether each chunk in the list is a phrase useful for representing conceptual content of documents. The inventive phrase recognition method uses the frequency information of two types of the partitioned text chunks (namely, the proper names in block **330** and the lower case phrases in blocks **335** and **345**) to make final phrase selection decisions. Preferably, the invention focuses on lower case phrases of more than one word, or on proper names ("John Hancock", "United States").

Referring to FIGS. **4B** and **5**, at block **201**, entries consisting of a solitary lower case word are not selected as phrases. Rejected entries are stored in memory block **320**.

As shown at block **202**, those chunks that include plural lower case words are determined to be phrases only if they occur at least twice in the text stream. These chunks are stored in memory block **335**. Chunks not fitting these criteria are stored in block **345** for further processing.

For chunks consisting of a solitary upper case word (either the first letter being capitalized or "allcaps"), no phrase decision is made at this stage, as shown at block **203**. Such chunks are stored in memory block **325**.

In block **204**, chunks including plural upper case words are determined to be proper names and are stored in a Proper Name List in memory block **330**.

Finally, other text chunks not fitting the previous criteria are simply discarded at this time, as indicated at block **205**.

Next, block **206** examines the lower case phrases having a single occurrence from memory block **245**. They are examined for having one of its sub-phrases as part of an existing lower case phrase in the list. For efficiency, a sub-phrase may be defined to be the first or last two or three words in the phrases. When the existence of a sub-phrase is detected, it is merged into the corresponding phrase in the list in memory block **335**, and its frequency count is updated. Otherwise, and the lower case phrase is decomposed into individual words for updating the lower case word list in memory block **320** as an optional output.

As a result of this sub-phrase mapping in block **206**, in our example the list is reduced to a list of lower case phrases and a list of proper names, both with their respective frequency counts:

[political wing, 2]
[Citing, 1]
[Irish Republican Army, 1]
[Clinton Administration, 1]
[Northern Ireland, 1]
[Manhattan, 1]
[Tuesday, 1]
[Administration, 1]
[Gerry Adams, 1]
[Sinn Fein, 1]
[White House, 1]
[British Government, 1]

The singleton upper case word could be used for referencing an existing proper name in the proper name list. To make the final frequency count accurate, the method makes one additional scan to the Proper Name List **330**. It consolidates the upper case word that is either the first or the last word of a previously recognized proper name, and updates its frequency count. This use of upper case single words in memory block **325** to revise the Proper Name List **330** is indicated at block **207**. The method stores the other upper case words in the upper case word list **325** as an optional output.

A special case of the singleton upper case word is that of the acronym. An acronym is defined either as a string of the first character of each word (which is neither a preposition nor a conjunction) in a proper name or as a string of the first character of each word in a proper name followed by a period. As indicated at block **208**, when an acronym in memory block **325** maps to a proper name in the proper name list **330**, the frequency count of the proper name is incremented, and the pair of the proper name and its acronym is copied into an acronym list **350** as an optional output.

In our example, this reference checking process further reduces the proper name list in this example to the following:

[Irish Republican Army, 1]
[Clinton Administration, 2]
[Northern Ireland, 1]
[Gerry Adams, 1]
[Sinn Fein, 1]
[White House, 1]
[British Government, 1]

If no additional processing is necessary, this method concludes by combining the lower case phrase list in memory block **335** and the Proper Name List in memory block **330** into a single Phrase List **340** which is provided as the final output of the phrase selection process **20**.

In another embodiment, the lower case phrases with frequency=1 in memory block **345** are also included in the consolidation, in addition to the Proper Name List in memory block **330** and the lower case phrases having frequency greater than 1 in memory block **335**. The choice of either including or excluding the lower case phrases in memory block **345** is determined by a frequency threshold parameter which determines the number of times a lower case phrase must be encountered before it is allowed to be consolidated into the final Phrase List.

The example shown in FIG. **5** has this threshold set to 2, so that those phrases encountered only once (in memory block **345**) are excluded from the consolidated Phrase List **340**. The dotted line extending downward from Phrase List **340** to include memory block **345** shows how lower case phrases encountered only once can be included in the Phrase List if desired, however.

In any event, the consolidation of memory blocks into a single Phrase List is indicated at block **209**.

For this text stream example, the final Phrase List is as follows:

[political wing, 2]
[Irish Republican Army, 1]
[Clinton Administration, 2]
[Northern Ireland, 1]
[Gerry Adams, 1]
[Sinn Fein, 1]
[White House, 1]
[British Government, 1]

The invention envisions that optional processes are available for further enhancing the recognized phrases.

FIG. **6A** is a flow chart of an optional processing module for consolidating with a synonym thesaurus.

Referring to FIG. **6A**, the Phrase List can be further reduced with a synonym thesaurus, as indicated at block **301**. The synonym thesaurus may be any suitable synonym

thesaurus available from commercial vendors. As an example, the Phrase List may map "White House" to "Clinton Administration." Using a synonym thesaurus is risky because its contents may not reflect the intended conceptual content of the text, and therefore may cause mapping errors. For example, it would be problematic if a synonym thesaurus maps "Bill Clinton" to "White House", because the two terms are not always equivalent.

FIG. 6B is a flow chart of an optional processing module for processing phrases with prepositions.

Referring to FIG. 6B, when a desirable lower case phrase contains one of a small set of prepositions (e.g., "right to counsel", "standard of proof"), the method takes the set out of the Partition List used for generating text chunks so that the phrase including the preposition has an opportunity to reveal itself as being part of a "good" phrase. This process is indicated as block **302**.

Since it is statistically unlikely that any given occurrence of a preposition is in a "good" phrase, this optional process consumes substantial time for a relatively small increase in phrases, and is considered optional.

It is necessary to have another process to further examine the unqualified phrase in memory block **345** that contains one of the selected prepositions, whether the sub-phrase on the left of the preposition or the sub-phrase on the right constitutes a valid phrase in the lower case phrase list in memory block **335**. This process is illustrated as block **303**.

As a result of process blocks **302, 303**, memory block **335** may be updated.

FIG. 6C is a flow chart of an optional processing module for trimming phrases with their collection frequencies.

Referring to FIG. 6C, still another optional process is that of editing the list of the Proper Name List **330** and lower case phrases **335** with additional frequency information **380** gathered from a text collection of more than one document. The assumption here is that, the more authors which use a phrase, the more reliable the phrase is for uniquely expressing a concept. In other words, a phrase occurring in more than one document is a "stronger" phrase than another phrase occurring only in a single document.

Here, the "collection frequency" of a phrase is the number of documents that contain the phrase. A collection frequency threshold (e.g., five documents) can be set to trim down those phrases whose collection frequencies are below the threshold, as indicated at block **304**. Essentially, FIG. 6C trims the entire Phrase List **340**, including entries from either memory block **330** or **335**.

When collection frequency information is available (as illustrated by memory block **380**), the minimum frequency requirement of two encounters for the lower case phrases within a text (see FIG. **5**) can be lowered to one encounter. "Mistaken" phrases will be rejected when consulting the collection frequency information when considering multiple documents.

FIG. 6D is a flow chart of an optional processing module for categorizing proper names.

Referring now to FIG. 6D, after proper names are identified and are stored in the Proper Name List **330**, it is possible to categorize them into new sets of pre-defined groups, such as company names, geographic names, organization names, peoples' names, and product names.

A list **385A** of company indicators (e.g., "Co." and "Limited") is used for determining whether the last word in a proper name is such an indicator, and thereafter for categorizing it into the group of company name. Any word after this indicator is removed from the proper name.

With the knowledge of the company name, it may be useful to check the existence of the same company name in the list that does not have the indicator word. If the search is successful, the frequency count of the company name is updated. The recognized company names are kept in a Company Names list **390A** as an optional output, as indicated at block **305**.

Similarly, a list **385B** of geographic names or a list **385C** of product names may be used for looking up whether a proper name has a match and thereafter for categorizing it into the list of geographic names or a list of product names, respectively. The recognized geographic names or product names are kept in Geographic Location Names **390B** or Product Names **390C** lists as optional outputs, as indicated at blocks **306** and **307**.

A list **385D** of words that designate organizations is used for determining whether the first or the last word of a proper name is the indicator of organization, and thereafter for categorizing it into the group of organizations. The recognized organization names may be kept in an Organization Names List **390D** as an optional output, as indicated at block **308**.

Finally a list **385E** of English first names is used for determining whether the first word of a proper name is a popular first name and thereafter for categorizing it into the group of peoples' names. Any word before the first name is removed from the proper name. The more comprehensive the lists are, the more people names can be categorized properly. The recognized people names are kept in a separate English Names list **390E** as an optional output for other indexing activities, as indicated at block **309**.

Appendices A through E present an exemplary Partition List **310** and exemplary Special Indicator/Name lists **385A–385E**.

The inventive method having been described above, the invention also encompasses apparatus (especially programmable computers) for carrying out phrase recognition. Further, the invention encompasses articles of manufacture, specifically, computer readable memory on which the computer-readable code embodying the method may be stored, so that, when the code used in conjunction with a computer, the computer can carry out phrase recognition.

Non-limiting, illustrative examples of apparatus which invention envisions are described above and illustrated in FIGS. **1** and **2**. Each constitutes a computer or other programmable apparatus whose actions are directed by a computer program or other software.

Non-limiting, illustrative articles of manufacture (storage media with executable code) may include the disk memory **116** (FIG. **1**), the disk memories **52–54** (FIG. **2**), other magnetic disks, optical disks, conventional 3.5-inch, 1.44 MB "floppy" diskettes or other magnetic diskettes, magnetic tapes, and the like. Each constitutes a computer readable memory that can be used to direct the computer to function in a particular manner when used by the computer.

Those skilled in the art, given the preceding description of the inventive method, are readily capable of using knowledge of hardware, of operating systems and software platforms, of programming languages, and of storage media, to make and use apparatus for phrase recognition, as well as computer readable memory articles of manufacture which, when used in conjunction with a computer can carry out phrase recognition. Thus, the invention's scope includes not only the method itself, but apparatus and articles of manufacture.

Applications of the phrase recognition method. The phrase recognition method described above can be used in a variety of text searching systems. These include, but need not be limited to, dynamic phrase recognition in on-line

systems, batch phrase recognition in a distributed development system, and batch phrase recognition in a mainframe system. The following description of the applications of the inventive phrase recognition method is illustrative, and should not limit the scope of the invention as defined by the claims.

In an on-line system (OLS) envisioned as a target application for the inventive phrase recognition method, a user viewing a current document and entering a command to search for documents of similar conceptual content must wait for the phrase recognition process to be completed. Accordingly, the efficiency of the inventive phrase recognition method is important, as it allows reduced response time and uses minimal resources in a time-sharing environment.

According to the application of the invention in a given on-line system, the method processes the text in a single document in real time to arrive at a list of "good" phrases, namely, ones which can be used as accurate and meaningful indications of the document's conceptual content, and which can be used as similarly accurate and meaningful queries in subsequent search requests. In particular, according to a preferred application, the Phrase List derived from the single document is used to construct a new search description to retrieve additional documents with similar conceptual content to the first document.

This implementation of the phrase recognition method may, for example, be embedded in session administrator (FIG. 2) or other software which governs operation of the computer system on which the phrase recognition method. Of course, the particular implementation will vary with the software and hardware environment of the particular application in question.

FIG. 7A indicates a user's viewing of a document in accordance with a suitable on-line text search system, and invoking the inventive phrase recognition method to search for additional documents of similar conceptual content. In particular, block **701** assumes a user viewing a given document enters a command (such as ".more") to retrieve more documents similar in conceptual content to the current one being viewed.

When the ".more" command is entered, control passes to block **702** which indicates retrieval of the document being viewed and passing it to the session administrator or other software which includes the inventive phrase recognition software.

Block **703** indicates execution of the inventive phrase recognition method on the text in the retrieved document. A candidate phrase list is generated, based on that single document.

Block **704** indicates how the candidate phrase list generated from the single document may be validated against an existing (larger) phrase dictionary. The static phrase dictionary may be generated as described below, with reference to the batch phrase recognition application in a distributed development system.

If a candidate phrase does not already exist in the phrase dictionary, the candidate phrase is broken down into its component words. Ultimately, a list of surviving phrases is chosen, based on frequency of occurrence.

At decision block **705**, if at least a given threshold number of words or phrases (e.g., five words or phrases) is extracted, control passes from decision block **705** to block **708**, described below.

If, however, the given threshold number of words or phrases are not extracted, control passes from decision block **705** along path **706** back to block **701**, after displaying an error message at block **707** which indicates that the displayed current document could not successfully be processed under the ".more" command.

Block **708** indicates that the newly-added words or phrases are added to the search query which previously resulted in the user's viewing the current document. Block **709** indicates the system's displaying the new "combined" search query to the user. The user may edit the new query, or may simply accept the new query by pressing "enter".

FIG. 7B schematically indicates implementation of the phrase recognition method in a batch phrase recognition system in a distributed development system.

In contrast to the implementation of the on-line system of FIG. 7A, in the application shown in FIG. 7B, the phrase recognition method is applied to a large collection of documents, and produces a list of phrases associated with the entire collection. As mentioned above with reference to FIG. 7A, the phrase dictionary may be generated by this batch recognition process in the "distributed development domain" (DDD) when there is an abundance of idle system resources. When the on-line system then uses the resultant phrase dictionary, the phrase dictionary is essentially static, having been generated and modified outside the on-line sessions.

The FIG. 7B application takes longer to execute than the single-document phrase recognition process occurring in the dynamic phrase recognition in the on-line application. Accordingly, the FIG. 7B process is preferably executed as a batch process at times when overall system usage is not impaired, such as overnight. In particular, the software implementation of the phrase recognition/phrase dictionary building process may be implemented on SUN work stations.

As a background to FIG. 7B, a developer's control file defines which documents, and/or which portions of the documents, should be processed in a given run. Block **723** indicates a filtering process which filters out documents and portions of documents which are not desired to contribute to the phrase dictionary, based on the control file. Block **724** indicates application of the inventive phrase recognition method to the documents and portions of documents which have passed through filter process **723**.

The output of the phrase recognition process is a phrase list (PL) which, in the illustrated non-limiting embodiment, is stored as a standard UNIX text file on disk. In a preferred embodiment, single-word terms which are encountered are discarded, so that only multiple word phrases are included in the phrase list (PL).

For simplicity, each phrase is provided on a single line in the file. Block **725** indicates how the UNIX file is sorted using, for example, the standard UNIX sort utility, causing duplicate phrases to be grouped together. Block **725** also calculates the frequency of each of the grouped phrases.

If a given phrase occurs less than a given threshold number of times (e.g., five times as tested by decision block **727**) it is discarded, as indicated by decision block **726**. Only phrases which have been encountered at least that threshold number of times survive to be included in the revised Phrase List, as shown in block **728**.

The revised Phrase List is then transferred from the SUN work station to its desired destination for use in, for example, the on-line system described above. It may also be transferred to a main frame computer using a file transfer protocol FTP, to be processed by a phrase dictionary building program and compiled into a production phrase dictionary. This process is shown illustrated as block **729**.

Referring now to FIG. 7C, the application of the inventive phrase recognition method on a mainframe system is sche-

matically illustrated. In the illustrated application, the phrase recognition method is implemented as a batch process in a production mainframe. The process involves a random sample of documents from a larger collection of documents, and produces a set of phrases for each document processed. The process is preferably executed when system resources are not otherwise in high demand, such as overnight. The process of FIG. 7C is especially useful for use with statistical thesauri.

As a background, it is assumed that phrases may be considered to be "related" to each other if they occur in the same document. This "relationship" can be exploited for such purposes as expanding a user's search query. However, in order to provide this ability, large number of documents must first be processed.

Referring again to FIG. 7C, block **733** indicates the filtering of documents and portions thereof in accordance with specifications from a control file, in much the same manner as described with reference to FIG. 7B. Block **734** indicates the application of the inventive phrase recognition method to the documents which pass the filter. One set of terms (single words, phrases or both) is produced for each document and stored in respective suitably formatted data structure on a disk or other storage medium.

Further details of implementation of the applications of the inventive phrase recognition method depend on the particular hardware system, software platform, programming languages, and storage media being chosen, and lie within the ability of those skilled in the art.

The following List are exemplary, illustrative, non-limiting examples of a Partition List and other lists which may be used with an embodiment of the phrase recognition method according to the present invention.

List A
Example of PARTITION LIST
(On-Line System with News Data)
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

A	BAITH	EVERY-THING	HEREOF
A.M	BE	EXCEPT	HEREON
ABOUT	BECAME	FEB	HERETO
ABOVE	BECAUSE	FEBRUARY	HEREWITH
ACROSS	BEEN	FEW	HERN
AFFECT	BEFORE	FEWER	HERSELF
AFTER	BEING	FIFTEEN	HIC
AGAIN	BELOW	FIVE	HIM
AGO	BETWEEN	FOR	HIMSELF
ALL	BOTH	FOUR	HIS
ALREADY	BUT	FOURTEEN	HIS'N
ALSO	BY	FRI	HISSELF
ALTHOUGH	CAN	FRIDAY	HOC
ALWAY	COULD	FROM	HOW
AN	DEC	GET	HOWEVER
AND	DECEMBER	GO	I
ANOTHER	DID	GOT	I'D
ANY	DO	HAD	I'LL
ANYBODY	DOE	HAPPEN	I'M
ANYMORE	DUE	HARDLY	I'VE
ANYONE	DURING	HAS	I.E
ANYTHING	E.G	HAVE	IF
APR	EACH	HAVING	IN
APRIL	EIGHT	HE	INTO
ARE	EIGHTEEN	HENCE	IS
AROUND	EITHER	HER	IT
AS	ELEVEN	HER'N	ITS
ASIDE	EVEN	HERE	ITSELF
ASK	EVEN-TUALLY	HEREAFTER	JAN
AT	EVER	HEREBY	JANUARY
AUG	EVERYBODY	HEREIN	JUL
AUGUST	EVERYMAN	HEREIN-SOFAR	JULY

-continued

List A
Example of PARTITION LIST
(On-Line System with News Data)
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

AWAY	EVERYONE	AFTER	JUN
JUNE	ONLY	SOMEBODY	THREE
JUST	ONTO	SOMEONE	THROUGH
LIKE	OR	SOMETHING	THUR
MANY	OTHER	SOMETIME	THURSDAY
MAR	OTHERWISE	SOMEWHERE	THUS
MARCH	OUGHT	SOONER	THY
MAY	OUR	STILL	THYSELF
ME	OUR'N	SUCCUSSION	TILL
MIGHT	OURSELF	SUCH	TO
MINE	OURSELVE	SUN	TODAY
MON	OUT	SUNDAY	TOMORROW
MONDAY	OVER	TAKE	TOO
MORE	P.M	TEN	TUB
MUCH	PERHAP	THAE	TUESDAY
MUST	QUIBUS	THAN	TWELVE
MY	QUITE	THAT	TWENTY
MYSELF	RATHER	THE	TWO
N.S	REALLY	THEE	UN
NANE	REV	THEIR	UNDER
NEITHER	SAID	THEIRSELF	UNLESS
NEVERTHELESS	SAME	THEIRSELVE	UNTIL
NINE	SAT	THEM	UNTO
NINETEEN	SATURDAY	THEMSELVE	UP
NO	SAY	THEN	UPON
NOBLEWOMAN	SEE	THERE	US
NOBODY	SEEMED	THEREAFTER	USE
NONE	SELF	THEREBY	VERY
NOR	SEP	THEREFORE	VIZ
NOT	SEPT	THEREFROM	WAS
NOV	SEPTEMBER	THEREIN	WE
NOVEMBER	SEVEN	THEREOF	WED
NOW	SEVENTEEN	THEREON	WEDNESDAY
O.S	SEVERAL	THERETO	WERE
OCT	SHE	THEREWITH	WHAT
OCTOBER	SHOULD	THESE	WHATE'ER
OF	SINCE	THEY	WHATEVER
OFF	SIR	THIRTEEN	WHATSOE'ER
OFTEN	SIX	THIS	WHATSOEVER
ON	SIXTEEN	THOSE	WHEN
ONE	SO	THOU	WHENEVER
ONESELF	SOME	THOUGH	WHENSOEVER
WHERE	17TH		
WHEREBY	18TH		
WHEREEVER	19TH		
WHEREIN	2D		
WHETHER	2ND		
WHICH	20TH		
WHICHEVER	21ST		
WHICHSOEVER	3D		
WHILE	3RD		
WHO	4TH		
WHOEVER	5TH		
WHOM	6TH		
WHOMSOEVER	7TH		
WHOSE	8TH		
WHOSESOEVER	9TH		
WHOSO			
WHOSOEVER			
WHY			
WILL			
WITH			
WITHIN			
WITHOUT			
WOULD			
YA			
YE			
YES			
YESTERDAY			
YET			
YON			
YOU			

-continued

List A
Example of PARTITION LIST
(On-Line System with News Data)
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

YOUR
YOUR'N
YOURSELF
1ST
10TH
11TH
12TH
13TH
14TH
15TH
16TH

List B
Example of COMPANY INDICATOR LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

BROS
BROS.
BROTHERS
CHARTERED
CHTD
CHTD.
CL
CL.
CO
CO.
COMPANY
CORP
CORP.
CORPORATION
CP
CP.
ENTERPRISES
GP
GP.
GROUP
INC
INC.
INCORP
INCORP.
INCORPORATED
INE
INE.
LIMITED
LNC
LNC.
LTD
LTD.

List C
Example of PRODUCT NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

240sx	Infinity	Reebok
300sx	Ingres	Rolaids
4-Runner	JVC	SA8
7Up	Jaguar	Sable
Access	Jeep	Sentra
Adobe	Keds	Seven-Up
Altima	Kleenex	Solaris
Arid	L.O.C.	Sony
Avia	Lexus	Sprite
B-17	Linux	Suave
B17	Lotus	Sun
BMW	Magnavox	Sybase

-continued

List C
Example of PRODUCT NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

Bayer	Maxima	Taurus
Blazer	Mercedes	Tide
Bounty	Minolta	Toshiba
Camary	Mitsubishi	Tums
Cannon	Mustang	Tylenol
Chevy	Nike	Windex
Cirrus	Nikon	Windows
Coke	O52	Yashika
Converse	Oracle	Zoom
Corvette	P100	
Etonic	P120	
Excel	P133	
F-14	P60	
F-15	P75	
F-16	P90	
F-18	Paradox	
F-22	Pepsi	
F14	Preparation-H	
F15	Puffs	
F16	Puma	
F18	Quicken	
F22	Rave	

List D
Example of ORGANIZATION INDICATOR LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

ADMINISTRATION	MEN
AGENCY	ORGANIZATION
ARMY	PARLIAMENT
ASSEMBLY	PARLIMENT
ASSOCIATION	PARTY
BOARD	REPUBLIC
BUREAU	SCHOOL
CENTER	SENATE
CHURCH	SOCIETY
CLINIC	TEAM
CLUB	UNION
COLLEGE	UNIVERSITY
COMMISSION	
COMMITTEE	
COMMUNITY	
CONGRESS	
COUNCIL	
COURT	
CULT	
DEPARTMENT	
DEPT	
FACTION	
FEDERATION	
FOUNDATION	
GOVERNMENT	
GUILD	
HOSPITAL	
HOUSE	
INDUSTRY	
INSTITUTE	
LEAGUE	

List E
Example of ENGLISH FIRST-NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

AARON	ADOLF	ALBERT	ALLECIA
ABAGAIL	ADOLPH	ALBERTA	ALLEEN
ABBIE	ADOLPHUS	ALBIN	ALLEGRA

-continued

List E
Example of ENGLISH FIRST-NAME LIST

ABBY	ADORA	ALDO	ALLEN
ABE	ADRIAN	ALDUS	ALLENE
ABEGAIL	ADRIANE	ALEC	ALLICIA
ABEL	ADRIANNE	ALECIA	ALLIE
ABELARD	ADRIEN	ALECK	ALLISON
ABIGAIL	ADRIENNE	ALENE	ALLOYSIUS
ABNER	AERIEL	ALEX	ALLY
ABRAHAM	AGATHA	ALEXANDER	ALLYN
ABRAM	AGGIE	ALEXANDRA	ALMA
ACIE	AGGY	ALEXANDRINA	ALMETA
ACY	AGNES	ALEXEI	ALMIRA
ADA	AGNETA	ALEXI	ALMON
ADAH	AGUSTIN	ALEXIA	ALONZA
ADAIR	AHARON	ALEXIS	ALOYSIUS
ADALBERT	AIDA	ALF	ALPHA
ADALINE	AILEEN	ALFIE	ALPHONSUS
ADAM	AILEENE	ALFIO	ALTA
ADDAM	AILENE	ALFORD	ALTHEA
ADDIE	AIME	ALFRED	ALTON
ADDY	AIMEE	ALFREDA	ALVA
ADELA	AINSLEE	ALFY	ALVAH
ADELAIDE	AINSLEY	ALGERNON	ALVESTER
ADELBERT	AJA	ALICE	ALVIN
ADELE	AL	ALICIA	ALYCE
ADELENE	ALAIN	ALINE	AMALIA
ADELINE	ALAINE	ALISA	AMANDA
ADELLA	ALAN	ALISHA	AMARYLLIS
ADELLE	ALANAH	ALISON	AMBER
ADLAI	ALANNA	ALIX	AMBROSE
ADNA	ALASTAIR	ALLAN	AMBROSIA
AMBROSIUS	ANNEMARIE	ARLO	AUDREY
AMELIA	ANNETTA	ARMAND	AUDRIE
AMIE	ANNETTE	ARMIN	AUDRY
AMILE	ANNICE	ARMOND	AUDY
AMITY	ANNIE	ARNE	AUGIE
AMON	ANNINA	ARNETT	AUGUST
AMOS	ANNMARIE	ARNEY	AUGUSTINE
AMY	ANSEL	ARNIE	AUGUSTUS
ANA	ANSELM	ARNOLD	AURELIA
ANABEL	ANSON	ARON	AURELIUS
ANABELLE	ANTHONY	ART	AUSTEN
ANALISE	ANTOINE	ARTE	AUSTIN
ANASTASIA	ANTOINETTE	ARTEMIS	AUTHER
ANATOLY	ANTON	ARTEMUS	AUTRY
ANCIL	ANTONE	ARTHUR	AUVEL
ANDIE	ANTONETTE	ARTIE	AVA
ANDREA	ANTONI	ARTIS	AVERY
ANDREAS	ANTONIA	ARTY	AVIS
ANDREE	ANTONIO	ARVELL	AVITUS
ANDREI	ANTONY	ARVIE	AVON
ANDREJ	AP	ARVO	AVRAM
ANDREW	APOLLO	ARVON	AXEL
ANDY	APRIL	ASA	AZZIE
ANETTA	ARA	ASHELY	AZZY
ANETTE	ARAM	ASHER	BABETTE
ANGELA	ARBY	ASHLEIGH	BAILEY
ANGELICA	ARCH	ASHLEY	BAIRD
ANGELINA	ARCHIBALD	ASTER	BALTHAZAR
ANGELINE	ARCHIE	ASTOR	BAMBI
ANGELIQUE	ARETHA	ASTRID	BARB
ANGIE	ARIC	ATHENA	BARBARA
ANGUS	ARICA	ATHENE	BARBEE
ANITA	ARIEL	ATTILIO	BARBI
ANN	ARISTOTLE	AUBREY	BARBIE
ANNA	ARLAN	AUBRIE	BARBRA
ANNABEL	ARLEEN	AUBRY	BARNABAS
ANNABELLE	ARLEN	AUD	BARNABUS
ANNALEE	ARLENE	AUDEY	BARNABY
ANNE	ARLIE	AUDIE	BARNARD
ANNELIESE	ARLIN	AUDINE	BARNET
ANNELISE	ARLINE	AUDIO	BARNETT
BARNEY	BERRY	BOBBY	BROOKS
BARNY	BERT	BONME	BRUCE
BARRETT	BERTHA	BONNY	BRUNHILDA
BARRY	BERTHOLD	BOOKER	BRUNHILDE

-continued

List E
Example of ENGLISH FIRST-NAME LIST

BART	BERTON	BORIS	BRUNO
BARTHOLOMEW	BERTRAM	BOYCE	BRYAN
BARTON	BERTRAND	BOYD	BRYANT
BASIL	BERTRUM	BRACIE	BRYCE
BAYARD	BERYL	BRACK	BRYON
BEA	BESS	BRAD	BUBBA
BEATRICE	BESSIE	BRADDOCK	BUCK
BEATRIX	BETH	BRADLEY	BUCKY
BEAUREGARD	BETSEY	BRADLY	BUD
BEBE	BETSIE	BRAINARD	BUDDIE
BECCA	BETSY	BRAINERD	BUDDY
BECKY	BETTE	BRANDI	BUEL
BEE	BETTIE	BRANDY	BUELL
BELINDA	BETTY	BRANKA	BUFFIE
BELLA	BETTYE	BREK	BUFFY
BELLE	BEULAH	BRENARD	BUFORD
BEN	BEVERLEE	BRENDA	BUNNIE
BENEDICT	BEVERLIE	BRENDAN	BUNNY
BENJAMIN	BEVERLY	BRENT	BURL
BENJI	BEWANDA	BRET	BURNELL
BENNETT	BIFF	BRETT	BURNETTA
BENNIE	BILL	BRIAN	BURNICE
BENNO	BILLIE	BRICE	BURREL
BENNY	BILLY	BRIDGET	BURT
BENTLEY	BIRD	BRIDGETT	BURTON
BERKE	BJARNE	BRIDGETTE	BURTRAM
BERKELEY	BJORN	BRIDIE	BUSTER
BERKELY	BJORNE	BRIGIT	BUTCH
BERKLEY	BLAINE	BRIGITTE	BYRON
BERLE	BLAIR	BRIJITTE	CAESAR
BERNARD	BLAKE	BRITNY	CAITLIN
BERNETTA	BLANCA	BRITTANY	CAL
BERNETTE	BLANCH	BRITTNEY	CALE
BERNHARD	BLANCHE	BRITTNY	CALEB
BERNICE	BOB	BROCK	CALLA
BERNIE	BOBBI	BRODERICK	CALLIE
BERRIE	BOBBIE	BROOKE	CALLY
CALVIN	CARROLL	CHARLEEN	CHRYSTAL
CAM	CARSON	CHARLENE	CHUCK
CAMDEN	CARY	CHARLES	CHUMLEY
CAMERON	CARYL	CHARLESE	CICELY
CAMILE	CARYN	CHARLETON	CICILY
CAMILLA	CAS	CHARLEY	CINDI
CAMILLE	CASEY	CHARLIE	CINDY
CANDACE	CASI	CHARLINE	CLAIR
CANDI	CASPAR	CHARLISE	CLAIRE
CANDICE	CASPER	CHARLOTTA	CLARA
CANDIS	CASS	CHARLOTTE	CLARA-BELLE
CANDUS	CASSANDRA	CHARLTON	CLARE
CANDY	CASSIE	CHAS	CLARENCE
CANNIE	CASSIUS	CHASTITY	CLARICE
CARA	CATHARINE	CHAUNCEY	CLARINA
CAREN	CATHERINE	CHELSIE	CLARISSA
CAREY	CATHLEEN	CHER	CLARK
CARI	CATHLENE	CHERI	CLASSIE
CARIN	CATHRINE	CHERIE	CLAUD
CARL	CATHRYN	CHERYL	CLAUDE
CARLA	CATHY	CHESTER	CLAUDELLE
CARLEEN	CEASAR	CHET	CLAUDETTE
CARLENE	CEATRICE	CHIP	CLAUDIA
CARLETON	CECELIA	CHLOE	CLAUDINE
CARLINE	CECIL	CHLORIS	CLAUDIUS
CARLISLE	CECILE	CHRIS	CLAUS
CARLTON	CECILIA	CHRISSIE	CLAY
CARLY	CECILY	CHRISSY	CLAYMON
CARLYLE	CEDRIC	CHRISTA	CLAYTON
CARMINE	CEFERINO	CHRISTABEL	CLEIO
CAROL	CELESTE	CHRISTABELLE	CLEM
CAROLA	CELESTINE	CHRISTAL	CLEMENT
CAROLANN	CELIA	CHRISTIAAN	CLEMENTINE
CAROLE	CELINA	CHRISTIAN	CLEMENZA
CAROLE	CESAR	CHRISTIE	CLENELL
CAROLINE	CHAD	CHRISTINE	CLEO
CAROLYN	CHADWICK	CHRISTOFER	CLEOPHUS

-continued

List E
Example of ENGLISH FIRST-NAME LIST

CAROLYNN	CHAIM	CHRISTOPH	CLEOTHA
CARREN	CHANCY	CHRISTOPHER	CLEOTIS
CARRIE	CHANDLER	CHRISTOS	CLETA
CARRIN	CHARITY	CHRISTY	CLETUS
CLEVE	CORKY	DAGMAR	DARRIN
CLEVELAND	CORNEAL	DAGWOOD	DARRYL
CLEVON	CORNELIA	DAISEY	DARWIN
CLIFF	CORNELIUS	DAISY	DARYL
CLIFFORD	CORRIE	DALE	DASHA
CLIFT	CORRINE	DALTON	DAVE
CLIFTON	CORRINNE	DAMIAN	DAVEY
CLINT	CORRY	DAMIEN	DAVID
CLINTON	CORTNEY	DAMION	DAVIDA
CLIO	CORY	DAMON	DAVIE
CLITUS	COSMO	DAN	DAVY
CLIVE	COUNTEE	DAN'L	DAWN
CLOVIA	COURTLAND	DANA	DEAN
CLOVIS	COURTNEY	DANIEL	DEANDRA
CLOYD	COY	DANIELLA	DEANE
CLYDE	CRAIG	DANIELLE	DEANNA
CODELL	CRIS	DANNA	DEANNE
COLBERT	CRISPIN	DANNY	DEB
COLE	CRISPUS	DANO	DEBBI
COLEEN	CRISSIE	DANUTA	DEBBIE
COLETTE	CRISSY	DAPHNE	DEBBY
COLIN	CRISTABEL	DARBIE	DEBORA
COLITA	CRISTA-BELLE	DARBY	DEBORAH
COLLEEN	CRYSTAL	DARCEE	DEBRA
COLLETTE	CURLESS	DARCEY	DEBS
COLLIN	CURLY	DARCI	DEDIE
COLON	CURT	DARCIE	DEE
CONNIE	CURTIS	DARCY	DEEANN
CONNY	CY	DARIEN	DEEANNE
CONRAD	CYBIL	DARIO	DEEDEE
CONROY	CYBILL	DARIUS	DEIDRE
CONSTANCE	CYNDI	DARLA	DEL
CONSTANTIA	CYNDY	DARLEEN	DELAINE
COOKIE	CYNTHIA	DARLENE	DELANE
CORA	CYRIL	DARLINE	DELANO
CORABELLE	CYRILL	DARLYNE	DELBERT
CORDELIA	CYRILLA	DARNELL	DELIA
COREY	CYRUS	DAROLD	DELL
CORINE	DABNEY	DARREL	DELLA
CORINNE	DACIA	DARRELL	DELLO
CORKIE	DACIE	DARREN	DELMA
DELMAR	DEVORAH	DONELL	DUAIN
DELMAS	DEWANE	DONELLE	DUAINE
DELMO	DEWAYNE	DONICE	DUANE
DELNO	DEWEY	DONIS	DUB
DELORES	DEWITT	DONNA	DUDLEY
DELORIS	DEXTER	DONNELL	DUEL
DELOY	DEZ	DONNELLE	DUELL
DELTA	DIAHANN	DONME	DUFF
DEMETRICE	DIANA	DONNY	DUFFY
DEMETRIUS	DIANE	DONOVAN	DUGALD
DENARD	DIANNA	DORA	DUKE
DENE	DIANNE	DORCAS	DULCIE
DENICE	DICK	DORCE	DULSA
DENILLE	DICKEY	DOREEN	DUNCAN
DENIS	DICKIE	DORI	DURWARD
DENISE	DIDI	DORIAN	DURWOOD
DENNIE	DIEDRE	DORIE	DUSTIN
DENNIS	DIERDRE	DORIENNE	DUSTY
DENNY	DIETER	DORINE	DWAIN
DENNYS	DIETRICH	DORIS	DWAINE
DENORRIS	DIMITRI	DOROTHA	DWAYNE
DEO	DINA	DOROTHEA	DWIGHT
DEON	DINAH	DOROTHY	DYLAN
DEREK	DINO	DORRANCE	DYNAH
DEREWOOD	DION	DORRIS	EARL
DERICK	DIRK	DORSEY	EARLE
DERL	DIXIE	DORTHIE	EARLENE
DERMOT	DMITRI	DORTHY	EARLINE
DERMOTT	DOLLIE	DOSHIE	EARNEST

-continued

List E
Example of ENGLISH FIRST-NAME LIST

DERRALL	DOLLY	DOT	EARNESTINE
DERREK	DOLORES	DOTTIE	EARTHA
DERRICK	DOM	DOTTY	EARTHEL
DERRY	DOMENIC	DOTY	EBEN
DERWOOD	DOMENICK	DOUG	EBENEEZER
DESDEMONA	DOMER	DOUGIE	EBENEZER
DESI	DOMINIC	DOUGLAS	EBERHARD
DESIRE	DOMINICK	DOUGLASS	EBONY
DESIREE	DOMINICKA	DOY	ED
DESMOND	DOMINIQUE	DOYLE	EDD
DESMUND	DON	DREW	EDDIE
DEVON	DONALD	DRU	EDDY
EDGAR	ELEONORA	ELSIE	EPHRIAM
EDGER	ELI	ELTON	ERASMUS
EDIE	ELIAS	ELVA	ERBIN
EDISON	ELIC	ELVERT	ERIC
EDITA	ELIJAH	ELVIE	ERICA
EDITH	ELINORE	ELVIN	ERICH
EDMOND	ELIOT	ELVIRA	ERICK
EDMUND	ELISABETH	ELVIS	ERIK
EDNA	ELISE	ELVON	ERIKA
EDRIS	ELISHA	ELWOOD	ERIN
EDSEL	ELISSA	ELY	ERLAND
EDUARD	ELIUS	ELZA	ERLE
EDWARD	ELIZA	EMELDA	ERMA
EDWIN	ELIZABETH	EMERIC	ERMAN
EDWINA	ELIZAR	EMERY	ERNEST
EDY	ELKE	EMETT	ERNESTINE
EDYTH	ELLA	EMIL	ERNESTO
EDYTHE	ELLE	EMILE	ERNIE
EFFIE	ELLEN	EMILIE	ERNST
EFRAIM	ELLERY	EMILY	ERROL
EGBERT	ELLIE	EMMA	ERVAN
EGIDIO	ELLIET	EMMALINE	ERVEN
EILEEN	ELLIOT	EMMERY	ERVIN
ELA	ELLIOTT	EMMET	ERWIN
ELAINE	ELLIS	EMMETT	ESAU
ELAYNE	ELLY	EMMIE	ESMERELDA
ELBA	ELLYN	EMMOT	ESTA
ELBERT	ELMA	EMMOTT	ESTEL
ELBERTA	ELMER	EMMY	ESTELA
ELDA	ELMIRA	EMOGENE	ESTELLA
ELDINE	ELMO	EMORY	ESTELLE
ELDON	ELNOR	ENDORA	ESTHR
ELDORA	ELNORA	ENDRE	ESTHA
ELE	ELOI	ENGBLBERT	ESTHER
ELEANOR	ELOISE	ENID	ETHAN
ELEANORA	ELOUISE	ENISE	ETHEL
ELEANORE	ELOY	ENNIS	ETHELENE
ELECTRA	ELRIC	ENOCH	ETHELINE
ELENA	ELROY	ENOLA	ETHYL
ELENORA	ELSA	ENZO	ETIENNE
ELENORE	ELSBETH	EPHRAIM	ETTA
ETTIE	FARRELL	FRANCIA	GALE
EUDORA	FARRIS	FRANCINE	GALLON
EUFA	FATIMA	FRANCIS	GARETH
EUGENE	FAUN	FRANCOIS	GARLAND
EUGENIA	FAWN	FRANCOISE	GARNET
EUGENIE	FAY	FRANK	GAROLD
EUGENIO	FAYE	FRANKIE	GARRET
EULA	FELECIA	FRANKLIN	GARRETT
EULALEE	FELICIA	FRANKLYN	GARRIE
EULALIE	FELICITY	FRANKY	GARRY
EULOGIO	FELIX	FRANNIE	GARTH
EUNACE	FELIZ	FRANNY	GARVIN
EUNICE	FERD	FRANZ	GARY
EUPHEMIA	FERDINAND	FRANZI	GASTON
EUSTACE	FERGUS	FRANZIE	GAVIN
EVA	FERN	FRANZY	GAY
EVALEE	FERREL	FRED	GAYE
EVAN	FERRELL	FREDA	GAYLE
EVANDER	FERRIS	FREDDIE	GAYLORD
EVANGELINE	FIDELE	FREDDY	GEARY
EVE	FIFI	FREDERICH	GEMMA
EVELYN	FILBERT	FREDERICK	GENA

List E
Example of ENGLISH FIRST-NAME LIST

EVERETT	FILIPPO	FREDERIK	GENE
EVERETTE	FIONA	FREEMAN	GENEVA
EVETTE	FITZ	FREIDA	GENEVIEVE
EVITA	FLETCHER	FRIEDA	GENIE
EWALD	FLO	FRIEDRICH	GENNARO
EWIN	FLOR	FRITZ	GENNIE
EZEKIEL	FLORA	FRONA	GENNIFER
EZRA	FLORANCE	FYODOR	GENNY
FABIAN	FLORENCE	GABBEY	GENO
FABIEN	FLORIDA	GABBIE	GEO
FAIRLEIGH	FLOSSIE	GABE	GEOFF
FAITH	FLOSSY	GABRIEL	GEOFFREY
FANNIE	FLOYD	GABRIELE	GEORGE
FANNY	FONDA	GABRIELLE	GEORGES
FARAH	FONTAINE	GABY	GEORGETTE
FARLEIGH	FORD	GAEL	GEORGIA
FARLEY	FORREST	GAETANO	GEORGIE
FARRAH	FRAN	GAGE	GEORGINA
FARREL	FRANCES	GAIL	GERALD
GERALDINE	GIULIA	GREGGORY	GWENETTA
GERARD	GIUSEPPE	GREGORY	GWENETTE
GERD	GIUSEPPI	GRETA	GWENITH
GERDINE	GIUSEPPINA	GRETCHEN	GWENN
GERHARD	GLADIS	GRETEL	GWENNE
GERI	GLADYCE	GRIFF	GWENNETH
GERMAIN	GLADYS	GRIFFIN	GWENYTH
GERMAINE	GLEN	GRIFFITH	GWYEN
GEROLD	GLENDA	GROVER	GWYLLEN
GEROME	GLENDON	GUENTHER	GWYN
GERRIE	GLENN	GUERINO	GWYNETH
GERRIT	GLENNA	GUIDO	GWYNNE
GERRY	GLENME	GUILLERMINA	GYULA
GERT	GLENNIS	GUILLERMO	HAILEY
GERTA	GLENNON	GUISEPPE	HAL
GERTIE	GLORIA	GUNNER	HALLIE
GERTRUDE	GLYN	GUNTER	HALLY
GEZA	GLYNDA	GUNTHER	HAMISH
GIA	GLYMS	GUS	HAMPTON
GIACOMO	GLYNN	GUSSIE	HANK
GIDEON	GLYNMS	GUSSY	HANNA
GIFFORD	GODFREY	GUST	HANNAH
GIGI	GODFRY	GUSTAF	HANNE
GIL	GODWIN	GUSTAV	HANNES
GILBERT	GOLDA	GUSTAVE	HANNIBAL
GILDA	GOLDIE	GUSTAVUS	HANS
GILES	GOLDY	GUSTOV	HANSELL
GILLIAN	GOMER	GUY	HARLAN
GILLIGAN	GORDAN	GWEN	HARLEN
GINA	GORDON	GWENDA	HARLEY
GINGER	GOTTFRIED	GWENDALYN	HARLIE
GINNI	GRACE	GWENDEN	HARLY
GINNIE	GRACIA	GWENDLYN	HARMIE
GINNY	GRACIE	GWENDOLA	HARMON
GINO	GRADY	GWENDOLEN	HAROL
GIORA	GRAEME	GWENDOLINE	HAROLD
GIOVANNA	GRAHAM	GWENDOLY	HARREL
GIOVANNI	GRANT	GWENDOLYN	HARRIET
GISELA	GRAYCE	GWENDOLYNE	HARRIETT
GISELLA	GREG	GWENDY	HARRIETTA
GISELLE	GREGG	GWENETH	HARRIS
HARRISON	HENRYK	HOLLIS	ILA
HARROLD	HERE	HOLLISTER	ILAH
HARRY	HERBERT	HOLLY	ILEEN
HARVEY	HERBIE	HOLLYANN	ILENA
HARVIE	HERBY	HOLLYANNE	ILENE
HASSLE	HERCULE	HOMER	ILSE
HATTIE	HERLINDA	HONEY	IMELDA
HATTY	HERM	HONORIA	IMOGENE
HAYDEE	HERMA	HOPE	INA
HAYDEN	HERMAN	HORACE	INGA
HAYDON	HERMANN	HORST	INGE
HAYLEY	HERMIE	HORTENSE	INGGA
HAYWOOD	HEROLD	HORTENSIA	INGRAM
HAZEL	HERSCH	HORTON	INGRID
HEATHCLIFF	HERSCHEL	HOSBA	IOLA

List E
Example of ENGLISH FIRST-NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

HEATHER	HERSHEL	HOWARD	IONA
HECTOR	HESTER	HOWIE	IONE
HEDDA	HESTHER	HOYT	IRA
HEDDIE	HETTA	HUBERT	IRAD
HEDWIG	HETTIE	HUEL	IRENA
HEIDI	HETTLE	HUEY	IRENE
HEINRICH	HETTY	HUGH	IRIS
HEINZ	HEYWOOD	HUGO	IRL
HELAINE	HEZEKIAH	HULDA	IRMA
HELEN	HILARY	HULDAH	IRVIN
HELENA	HILDA	HUMPHREY	IRVING
HELENE	HILDE	HUNTINGTON	IRWIN
HELGA	HILDEGARD	HURSCHEL	ISAAC
HELMUT	HILDE-GARDE	HY	ISAAK
HELMUTH	HILDRED	HYACINTH	ISABEL
HELOISE	HILDUR	HYMAN	ISABELLA
HENDRIK	HILLARD	IAIN	ISABELLE
HENDRIKA	HILLARY	IAN	ISAC
HENNIE	HILLERY	ICHABOD	ISADOR
HENNY	HIRAM	IDA	ISADORA
HENRI	HIRSCH	IGGY	ISADORE
HENRIETA	HOBART	IGNATIUS	ISMAH
HENRIETTA	HOBERT	IGNATZ	ISHAM
HENRIETTE	HOLLEY	IGNAZ	ISHMAEL
HENRIK	HOLLI	IGOR	ISIAH
HENRY	HOLLIE	IKE	ISIDOR
ISIDORE	JAN	JEMIMA	JILLIAN
ISMAEL	JANA	JEMMA	JIM
ISOM	JANE	JEMMIE	JIMBO
ISRAEL	JANEEN	JEMMY	JIMME
ISTVAN	JANEL	JENNIE	JIMMIE
ITA	JANELL	JENNIFER	JIMMY
IVA	JANELLE	JENNY	JINNY
IVAN	JANET	JENS	JO
IVANA	JANICE	JERALD	JOAB
IVAR	JANIE	JERALDINE	JOACHIM
IVARS	JANINE	JERE	JOAN
IVETTE	JANIS	JEREMIAH	JOANN
IVEY	JANISE	JEREMIAS	JOANNA
IVIE	JANNA	JEREMY	JOANNE
IVO	JAQUELYN	JERI	JOB
IVONNE	JARED	JERIANE	JOCELYN
IVOR	JARRETT	JERIE	JOCILYN
IVORY	JARRYL	JERMAIN	JOCLYN
IVY	JARVIS	JERMAINE	JODI
IZAAK	JAS	JEROL	JODIE
IZZIE	JASMINE	JEROLD	JODY
IZZY	JASON	JEROME	JOE
JAC	JASPER	JERRALD	JOEL
JACK	JAY	JERRED	JOETTE
JACKI	JAYME	JERRELD	JOEY
JACKIE	JAYMEE	JERRELL	JOFFRE
JACKSON	JAYNE	JERRI	JOH
JACKY	JEAN	JERRIE	JOHANN
JACOB	JEANETTE	JERROLD	JOHANNES
JACOBUS	JEANIE	JERRY	JOHN
JACQUELIN	JEANNE	JERZY	JOHNIE
JACQUELINE	JEANNETTE	JESS	JOHNNA
JACQUELYN	JEANNINE	JESSE	JOHNNIE
JACQUES	JEBEDIAH	JESSICA	JOHNNY
JADE	JED	JESSIE	JOJO
JAKE	JEDEDIAH	JETHRO	JOLENE
JAKOB	JEFEREY	JETTA	JON
JAMES	JEFF	JETTIE	JONAH
JAMEY	JEFFEREY	JEWEL	JONAS
JAMIE	JEFFIE	JEWELL	JONATHAN
JAMYE	JEFFREY	JILL	JONATHON
JONELL	JULIUS	KELSEY	KORNELIA
JOM	JUNE	KELVIN	KORNELIUS
JONNY	JUNEY	KEN	KRAIG
JORDAN	JUNIE	KENDALL	KRIS
JORDON	JUNIOR	KENDELL	KRISTA
JOSEA	JUNIUS	KENDRICK	KRISTEN
JOSELYN	JURGEN	KENNETH	KRISTI

-continued

List E
Example of ENGLISH FIRST-NAME LIST

JOSEPH	JUSTEN	KENNY	KRISTIE
JOSEPHA	JUSTICE	KENT	KRISTIN
JOSEPHINE	JUSTIN	KERI	KRISTINE
JOSEY	JUSTINE	KERMIT	KRISTOFFER
JOSH	KALVIN	KERRIE	KRISTY
JOSHUA	KARA	KERRY	KURT
JOSIAH	KAREN	KERSTIN	KYLE
JOSIE	KARIN	KERVIN	KYM
JOY	KARL	KERWIN	LACEY
JOYCE	KARLA	KEVIN	LACIE
JOYCEANN	KAROLA	KIERSTEN	LACY
JOZEF	KASEY	KILIAN	LADON
JOZSEF	KASPAR	KILLIAN	LAINE
JUBAL	KASPER	KIM	LALAH
JUD	KATE	KIMBER	LAMAR
JUDAH	KATERINA	KIMBERLEE	LAMARTINE
JUDAS	KATEY	KIMBERLEIGH	LAMBERT
JUDD	KATHERINE	KIMBERLEY	LAMONT
JUDE	KATHERYN	KIMBERLY	LANA
JUDI	KATHLEEN	KIP	LANCE
JUDIE	KATHRYN	KIRBY	LANCELOT
JUDITH	KATHY	KIRK	LANE
JUDY	KATIE	KIRSTEN	LANSON
JULEE	KATY	KIRSTI	LARA
JULES	KAY	KIRSTIE	LARENE
JULIA	KAYE	KIRSTIN	LARONE
JULIAN	KAYLEEN	KIRSTY	LARRIS
JULIANE	KEENAN	KIT	LARRY
JULIANNE	KEISHA	KITTIE	LARS
JULIE	KEITH	KITTY	LASKA
JULIEN	KELLEY	KLAUS	LASLO
JULIENNE	KELLI	KONRAD	LASZLO
JULIET	KELLIE	KONSTAN-TINOS	LATIC
JULIETTE	KELLY	KOOS	LATIMER
LAUANNA	LELAH	LEVON	LISE
LAUNCIE	LELAND	LEW	LISSA
LAURA	LELIA	LEWELL	LIZ
LAUREE	LELIO	LEWIS	LIZA
LAUREL	LEMMY	LEXIE	LIZABETH
LAUREN	LEMUEL	LEXY	LIZZIE
LAURENCE	LEN	LIANE	LIZZY
LAURETA	LENA	LIBBIE	LLEWELLYN
LAURETTA	LENARD	LIBBY	LLOYD
LAURETTE	LENDEN	LIDA	LLOYDA
LAURI	LENETTE	LIEF	LOELLA
LAURIE	LENISE	LILA	LOGAN
LAURIEN	LENNIE	LILAC	LOIS
LAVELL	LENNY	LILAH	LOLA
LAVERA	LENORA	LILE	LOLETA
LAVERN	LENORE	LILIAN	LOLITA
LAVERNA	LENWOOD	LILIEN	LOLLIE
LAVERNE	LEO	LILITH	LOLLY
LAVINA	LEOLA	LILLIA	LON
LAVIMA	LEON	LILLIAN	LOM
LAVONNE	LEONA	LILLIE	LONNA
LAWRENCE	LEONARD	LILLY	LONME
LDA	LEONID	LILY	LONNY
LEA	LEONIDA	LIN	LONSO
LEAH	LEONIDAS	LINCOLN	LONZIE
LEANDER	LEONORA	LINDA	LONZO
LEANE	LEOPOLD	LINDSAY	LONZY
LEANN	LERA	LINDSEY	LORA
LEANNE	LEROY	LINK	LORAIN
LEATHA	LES	LINNEA	LORAINE
LEE	LESLIE	LINNIE	LORANE
LEEANN	LESTER	LINNY	LORAY
LEEANNE	LETA	LINROY	LORAYNE
LEENA	LETHA	LINUS	LOREEN
LEESA	LETICIA	LINVAL	LOREN
LEFFEL	LETITIA	LINWOOD	LORENA
LEFTY	LETTIE	LINZIE	LORENE
LEIF	LETTY	LINZY	LORETA
LEIGH	LEVERNE	LIONEL	LORETTA
LEILA	LEVERT	LISA	LORI

-continued

List E
Example of ENGLISH FIRST-NAME LIST

LEITH	LEVI	LISABETH	LORIN
LORNA	LUIGI	MADELENE	MANSEL
LORNE	LUKE	MADELINE	MARABEL
LORRAINE	LULA	MADELYN	MARC
LORRAYNE	LULAH	MADGE	MARCEL
LOTHAR	LULU	MADIE	MARCELIN
LOTTIE	LUMMIE	MADONNA	MARCELL
LOU	LUNA	MAE	MARCELLA
LOUELLA	LURLEEN	MAGDA	MARCELLE
LOUIE	LURLENE	MAGDALENA	MARCELLUS
LOUIS	LURLINE	MAGDALENE	MARCI
LOUISA	LUTHER	MAGDALINE	MARCIA
LOUISE	LUZ	MAGGIE	MARCIE
LOULA	LUZERNE	MAGGY	MARCUS
LOURETTA	LYDIA	MAGNUS	MARCY
LOVELL	LYLE	MAHALA	MARDA
LOVETTA	LYMAN	MAHALIA	MARGARET
LOVETTE	LYN	MAIA	MARGE
LOWELL	LYNDA	MAIBLE	MARGEAUX
LOY	LYNDON	MAIJA	MARGERY
LOYAL	LYNN	MAJOR	MARGI
LUANN	LYNNA	MAL	MARGIE
LUANNA	LYNNE	MALCOLM	MARGO
LUANNE	LYNNETTE	MALINDA	MARGOT
LUBY	LYNWOOD	MALISSA	MARGRET
LUCAS	LYSANDER	MALKA	MARGY
LUCIAN	M'LINDA	MALLORIE	MARIAM
LUCIE	MABEL	MALLORY	MARIAN
LUCIEN	MABELLE	MALORIE	MARIANN
LUCILE	MABLE	MALORY	MARIANNE
LUCILLA	MAC	MALYNDA	MARIBEL
LUCILLE	MACE	MAME	MARIBELLE
LUCINDA	MACIE	MAMIE	MARIBETH
LUCIUS	MACK	MANDEE	MARIE
LUCRETIA	MADALAINE	MANDI	MARIEL
LUCY	MADALEINE	MANDIE	MARIETTA
LUDLOW	MADALINE	MANDY	MARIETTE
LUDWIG	MADALYN	MANFRED	MARILEE
LUDWIK	MADDIE	MANICE	MARILYN
LUDY	MADDY	MANLEY	MARILYNN
LUELLA	MADELAINE	MANLY	MARINA
LUGENE	MADELEINE	MANNY	MARIO
MARION	MARYBETH	MEAGAN	MEYER
MARISSA	MARY-FRANCES	MEG	MIA
MARJUS	MARYLOU	MEGAN	MIATTA
MARJORIE	MASON	MEL	MICAH
MARJORY	MATE	MELANIE	MICHAEL
MARK	MATEY	MELANY	MICHEL
MARKEE	MATHEW	MELBA	MICHELE
MARKIE	MATHIAS	MELICENT	MICHELLE
MARKOS	MATHILDA	MELINDA	MICK
MARKUS	MATILDA	MELISSA	MICKEY
MARLA	MATT	MELLICENT	MICKIE
MARLENA	MATTHEW	MELODI	MICKY
MARLENE	MATTHIAS	MELODIE	MIDGE
MARLEY	MATTIE	MELODY	MIKAEL
MARLIN	MATTY	MELONIE	MIKAL
MARLON	MATTYE	MELONY	MIKE
MARMADUKE	MAUD	MELVA	MIKEAL
MARNEY	MAUDE	MELVIN	MILAN
MARNIE	MAURA	MELVYN	MILDRED
MARNY	MAUREEN	MELYNDA	MILES
MARSDEN	MAURENE	MENDEL	MILICENT
MARSHA	MAUREY	MERCEDEL	MILLARD
MARSHAL	MAURICE	MERCEDES	MILLICENT
MARSHALL	MAURIE	MERCY	MILLIE
MARTA	MAURINE	MEREDETH	MILLY
MARTEA	MAURY	MEREDITH	MILO
MARTHA	MAVIS	MERIDETH	MILT
MARTI	MAX	MERIDITH	MILTON
MARTICA	MAXCIE	MERISSA	MIMI
MARTIE	MAXCINE	MERLE	MINDY
MARTIKA	MAXIE	MERLIN	MINERVA
MARTILDA	MAXIM	MERLYN	MINNIE

-continued

List E
Example of ENGLISH FIRST-NAME LIST

MARTIN	MAXI-MILIAN	MERREL	MIRANDA
MARTY	MAXI-MILLIAN	MERRELL	MIRIAM
MARV	MAXINE	MERRILL	MISSIE
MARVA	MAXWELL	MERRY	MISSY
MARVIN	MAY	MERVIN	MISTY
MARY	MAYBELLE	MERWIN	MITCH
MARYAM	MAYDA	MERWYN	MITCHEL
MARYANN	MAYME	MERYL	MITCHELL
MARYANNE	MAYNARD	META	MITZI
MOE	NAMIE	NEWEL	NORA
MOLLIE	NAN	NEWELL	NORAH
MOLLY	NANCI	NEWT	NORBERT
MONA	NANCIE	NEWTON	NOREEN
MONICA	NANCY	NICHAEL	NORM
MONIQUE	NANETTE	NICHOLM	NORMA
MONTE	NAM	NICHOLAS	NORMAL
MONTGOMERY	NANNETTE	NICK	NORMAN
MONTY	NANNI	NICKI	NORRIS
MONY	NANNIE	NICKIE	NORTON
MORDECAI	NANNY	NICKODEMUS	NORVAL
MOREY	NAOMA	NICKY	NUNZIO
MORGAN	NAOMI	NICODEMO	NYLE
MORGANA	NAPOLEON	NICODEMUS	OBADIAH
MORRIS	NAT	NICOL	OBED
MORT	NATALIE	NICOLA	OBEDIAH
MORTIMER	NATASHA	NICOLM	OCIE
MORTON	NATASSIA	NICOLAS	OCTAVE
MORTY	NATHAN	NICOLE	OCTAVIA
MOSE	NATHAMEL	NICOLETTE	ODEL
MOSES	NATWICK	NICOLLE	ODELL
MOZELLE	NAZARETH	NICOLO	ODESSA
MULLIGAN	NEAL	NIGEL	ODIE
MURIEL	NEALY	NIKA	ODIS
MURPHY	NED	NIKE	OGDEN
MURRAY	NEDINE	NIKI	OKTAVIA
MURRELL	NEIL	NIKITA	OLA
MURRY	NEILL	NIKITAS	OLAF
MYLES	NELDA	NIKKI	OLAN
MYNA	NELL	NILE	OLEG
MYRA	NELLE	NILES	OLEN
MYRAH	NELLIE	NILS	OLGA
MYRAL	NELLY	NIMROD	OLIN
MYREN	NELS	NINA	OLIVE
MYRNA	NELSON	NOAH	OLIVER
MYRON	NENA	NOE	OLIVIA
MYRTLE	NERO	NOEL	OLLEN
NADENE	NERSES	NOLA	OLLIE
NADIA	NESTOR	NOLAN	OLOF
NADINE	NETFIE	NONA	OMER
NADJA	NEVILLE	NONNIE	ONAL
ONEL	PAGE	PENNY	PRUDENCE
ONNIK	PAIGE	PER	PRUE
OPAL	PAM	PERCIVAL	PRUNELLA
OPEL	PAMELA	PERCY	QUEEN
OPHELIA	PANSY	PERRY	QUEENIE
OPRAH	PAOLO	PERSIS	QUEENY
ORA	PARL	PETE	QUENTIN
ORAL	PARNELL	PETER	QUINCY
ORAN	PARRY	PETRA	QUINNIE
OREN	PASCAL	PETRO	RACHEL
ORESTE	PASCHAL	PETROS	RACHELLE
ORIN	PASQUALE	PHEBE	RAE
ORLAN	PAT	PHEUM	RAIFORD
ORLEN	PATIENCE	PHIDIAS	RALPH
ORLIN	PATRICE	PHIL	RAMONA
ORLYN	PATRICIA	PHILBERT	RANDAL
ORPHA	PATRICK	PHILIP	RANDALL
ORSON	PATSY	PHILIPPE	RANDI
ORTON	PATTI	PHILLIP	RANDIE
ORTRUD	PATTIE	PHILO	RANDOLF
ORVAL	PATTY	PHINEAS	RANDOLPH
ORVID	PAUL	PHOEBE	RANDY
ORVIL	PAULA	PHYLLIS	RANSOM

-continued

List E
Example of ENGLISH FIRST-NAME LIST

ORVILLE	PAULETTE	PIA	RAPHAEL
OSBERT	PAULINA	PIER	RAQUEL
OSCAR	PAULINE	PIERCE	RAY
OSGOOD	PAVEL	PIERINA	RAYFORD
OSIE	PEARCE	PIERRE	RAYMOND
OSSIE	PEARL	PIERS	RBT
OSWALD	PEARLENE	PIETER	REAGAN
OTHA	PEARLIE	PIETRO	REBA
OTHIS	PEARLINE	PLATO	REBECCA
OTIS	PEARLY	POINDEXTER	RECTOR
OTTIS	PEDER	POLUE	REED
OTTO	PEG	POLLY	REGAN
OVE	PEGGI	PORTIA	REGGIE
OVETA	PEGGIE	PRECY	REGGY
OWEN	PEGGY	PRESTON	REGINA
OZZIE	PENELOPE	PRINCE	REGINALD
PADDIE	PENNI	PRINCESS	REGIS
PADDY	PENNIE	PRISCILLA	REID
REINHARDT	RIPLEY	RONNETTE	ROXIE
REINHOLD	RISA	RONNIE	ROXY
REMI	RITA	RONNY	ROY
REMO	RITCHIE	ROOSEVELT	ROYAL
REMUS	RITHA	RORY	ROYCE
RENA	ROB	ROSALEE	ROZALIA
RENATA	ROBBI	ROSALIA	ROZALIE
RENATE	ROBBIE	ROSALIE	RUBAN
RENE	ROBBIN	ROSALIND	RUBEN
RENEE	ROBBY	ROSALINDA	RUBENA
RETA	ROBERT	ROSALYN	RUBERT
RETHA	ROBERTA	ROSALYND	RUBEY
REUBAN	ROBIN	ROSAMOND	RUBIE
REUBEN	ROBINA	ROSAMONDE	RUBIN
REUBENA	ROBINETTE	ROSAMUND	RUBINA
REUBIN	ROBYN	ROSAMUNDE	RUBY
REUBINA	ROCCO	ROSANNA	RUBYE
REVA	ROCHELLE	ROSANNE	RUDDY
REX	ROCKY	ROSCOE	RUDOLF
REXFORD	ROD	ROSE	RUDOLPH
REY	RODDIE	ROSEANN	RUDY
REYNALD	RODERIC	ROSEANNE	RUE
REYNOLD	RODERICH	ROSEBUD	RUEBEN
RHEA	RODERICK	ROSELIN	RUFUS
RHINELANDER	RODGER	ROSELYN	RULOEF
RHODA	RODNEY	ROSEMARIE	RUPERT
RHONA	RODRICK	ROSEMARY	RUSS
RHONDA	ROGER	ROSEMUND	RUSSELL
RHYS	ROLAND	ROSEMUNDE	RUSTY
RICARD	ROLF	ROSENA	RUTH
RICH	ROLFE	ROSETTA	RUTHANNA
RICHARD	ROLLO	ROSIE	RUTHANNE
RICHELLE	ROMAN	ROSINA	RUTHIE
RICHIE	ROMEO	ROSLYN	RUTHLYN
RICK	ROMULUS	ROSS	RYAN
RICKEY	RON	ROSWELL	SABA
RICKI	RONA	ROSWITHA	SABINA
RICKIE	RONALD	ROULETTE	SABINE
RICKY	RONDA	ROWENA	SABRINA
RIKKI	RONDELL	ROWLAND	SADIE
RILEY	RONETTE	ROXANNE	SAL
SALLI	SHARI	SIDNEY	STACIE
SALLIE	SHARLENE	SIEGFRIED	STACY
SALLY	SHARON	SIG	STAN
SALLYE	SHARYN	SIGFRIED	STANISLAW
SALOMON	SHAUN	SIGMUND	STANLEY
SAM	SHAWN	SIGNE	STANLY
SAMANTHA	SHAWNA	SIGURD	STEFAN
SAMMIE	SHAYNE	SILAS	STEFFI
SAMMY	SHEARL	SILVIA	STELLA
SAMUAL	SHEBA	SILVIO	STEPHAN
SAMUEL	SHEENA	SIMEON	STEPHANIE
SANDI	SHEILA	SIMON	STEPHEN
SANDIE	SHELBY	SIMONE	STERLING
SANDRA	SHELDEN	SIMONNE	STEVE
SANDY	SHELDON	SISSIE	STEVEN
SANJA	SHELIA	SISSY	STEVIE

-continued

List E
Example of ENGLISH FIRST-NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

SARA	SHELLEY	SKEET	STEWART
SARAH	SHELLY	SKIPPIE	STU
SAUL	SHELTON	SKIPPY	STUART
SCHUYLER	SHERI	SKYLER	SUANN
SCOT	SHERIDAN	SLIM	SUANNE
SCOTT	SHERIE	SMEDLEY	SUE
SCOTTIE	SHERILYN	SOFIE	SUELLEN
SCOTTY	SHERL	SOL	SUMNER
SEAMUS	SHERLE	SOLOMAN	SUNNIE
SEAN	SHERLEE	SOLOMON	SUNNY
SEBASTIAN	SHERMAN	SONDRA	SUSAN
SEFERINO	SHERON	SONIA	SUSANA
SELDEN	SHERREE	SONJA	SUSANNA
SELENA	SHERRI	SONNY	SUSANNAH
SELENE	SHERRIE	SONYA	SUSANNE
SELINA	SHERRY	SPARKY	SUSETTE
SELMA	SHERWIN	SPENCE	SUZAN
SELMER	SHERYL	SPENCER	SUZANNE
SERENA	SHIRL	SPENSER	SUZELLEN
SERINA	SHIRLE	SPIRO	SUZETTE
SETH	SHIRLEE	SPIROS	SUZI
SEYMORE	SHIRLEY	SPYROS	SUZIE
SEYMOUR	SI	STACEY	SUZY
SHANE	SIBYL	STACI	SVEN
SHANNON	SID	STACIA	SWEN
SY	TERRI	THURSTAN	TREVOR
SYBIL	TERRILL	THURSTON	TREY
SYD	TERRY	TIBOR	TRICIA
SYDNEY	TERRYL	TIFFANY	TRILBY
SYLVAN	TESS	TILLIE	TRINA
SYLVANUS	TESSIE	TILLY	TRISH
SYLVENE	TESSY	TIM	TRISTAM
SYLVESTER	TEX	TIMMY	TRISTAN
SYLVIA	THAD	TIMO	TRIXIE
SYLVIE	THADDEUS	TIMOTHY	TRIXY
SYLVINA	THADEUS	TINA	TROY
TABATHA	THARON	TINO	TRUDIE
TABITHA	THEA	TIPHANIE	TRUDY
TAD	THEDA	TIPHANY	TWALA
TAFFY	THEIMA	TITO	TWILA
TALLULAH	THELMA	TITUS	TWYLA
TAMARA	THEO	TOBIAS	TYCHO
TAMI	THEOBALD	TOBY	TYCHUS
TAMMI	THEODIS	TODD	TYCUS
TAMMIE	THEODOR	TOLLIE	TYRONE
TAMMY	THEODORA	TOLLIVER	UDIE
TANCRED	THEODORE	TOLLY	UDY
TAMA	THEODORIS	TOM	ULRICH
TANJA	THEODOSIA	TOMMIE	ULYSSES
TANYA	THEONE	TOMMY	UNA
TASHA	THEORA	TONEY	URA
TATE	THEOTIS	TOM	URBAIN
TATIANA	THERESA	TOMA	URIAS
TAUBA	THERESIA	TONY	URSULA
TED	THERESSA	TONYA	VACHEL
TEDD	THOM	TOOTIE	VADA
TEDDY	THOMAS	TOVE	VAL
TEE	THOMASINA	TRACEE	VALDA
TELMA	THOR	TRACEY	VALENTIN
TENA	THORA	TRACI	VALENTINE
TENCH	THORE	TRACIE	VALENTINO
TERENCE	THORNTON	TRACY	VALERIA
TERESA	THORVALD	TRAVER	VALERIE
TERRANCE	THOS	TRAVIS	VALTER
TERRELL	THURMAN	TRENA	VANCE
TERRENCE	THURMOND	TRENT	VANDER
VANDY	VINCENT	WAYMAN	WINNIFRED
VANBSSA	VINNIE	WAYNE	WINNY
VAUGHAN	VINNY	WELDEN	WINONA
VAUGHN	VIOLA	WELDON	WINSLOW
VEDA	VIOLET	WELLS	WINSTON
VELLA	VIRACE	WENDEL	WINTHROP
VELMA	VIRDA	WENDELL	WINTON
VELVA	VIRGIE	WENDI	WM
VEOLA	VIRGIL	WENDY	WOLFGANG

-continued

List E
Example of ENGLISH FIRST-NAME LIST
Copyright 1995
LEXIS-NEXIS, a Division of Reed Elsevier Inc.

VERA	VIRGINIA	WERNER	WOODIE
VERDEEN	VIRGIMUS	WES	WOODROW
VERDELL	VITA	WESLEY	WOODRUFF
VERGA	VITINA	WESLIE	WOODY
VERGIL	VITINO	WESTLEY	WYATT
VERLIE	VITO	WHITNEY	WYLA
VERLIN	VITTORIA	WIDDIE	XAVIER
VERLON	VIVIAN	WILBER	XAVIERA
VERLYN	VIVIANNE	WILBERFORCE	YARDLEY
VERN	VIVIEN	WILBERT	YETTA
VERNA	VLAD	WILBUR	YOLANDA
VERNARD	VLADIMIR	WILDA	YOSEF
VERNE	VOL	WILEY	YVES
VERNEST	VON	WILFORD	YVETTB
VERNESTINE	VONDA	WILFRED	YVONNE
VERNICE	VONNA	WILHELM	ZACHARIA
VERNIE	WADE	WILHELMENA	ZACHARIAH
VERNON	WALDEMAR	WILHELMINA	ZACHARY
VERONICA	WALDO	WILHEMENA	ZACK
VERSA	WALDORF	WILHEMINA	ZALPH
VERSIE	WALLACE	WILLARD	ZANE
VI	WALLIE	WILLIAM	ZEB
VIC	WALLY	WILLIE	ZEBADIAH
VICKI	WALT	WILLIS	ZEBEDEE
VICKIE	WALTER	WILLMA	ZECHARIAH
VICKY	WANDA	WILLY	ZEF
VICTOR	WARD	WILMA	ZEFF
VICTORIA	WARREN	WILMAR	ZEKE
VIDAL	WASHING-TON	WILMOT	ZELDA
VIE	WAYLAN	WINFRED	ZELIA
VILMA	WAYLEN	WINIFRED	ZELIG
VINCE	WAYLON	WINNIE	ZELL
ZELLA			
ZELLE			
ZELMA			
ZENA			
ZENITH			
ZENO			
ZENOBIA			
ZENON			
ZEPHERY			
ZETA			
ZETTA			
ZEV			
ZILLA			
ZILLAH			
ZINA			
ZITA			
ZIVKO			
ZOE			
ZOLLIE			
ZOLLY			
ZORA			
ZULA			
ZYGMUND			
ZYGMUNT			

Modifications and variations of the above-described embodiments of the present invention are possible, as appreciated by those skilled in the art in light of the above teachings. As mentioned, any of a variety of hardware systems, memory organizations, software platforms, and programming languages may embody the present invention without departing from its spirit and scope. Moreover, countless variations of the Partition List, company indicators, product names, organization indicators, English first name list, and resulting Phrase Lists, and the like, may be employed or produced while remaining within the scope of the invention. It is therefore to be understood that, within the scope of the appended claims and their equivalents, the invention may be practiced otherwise than as specifically described.

What is claimed is:

1. A computer-implemented method of processing a stream of document text to form a list of phrases that are indicative of conceptual content of the document, the phrases being used as index terms and search query terms in full text document searching performed after the phrase list is formed, the method comprising:

partitioning the document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

selecting certain chunks as the phrases of the phrase list, based on frequencies of occurrence of the chunks within the stream of document text.

2. The method of claim 1, wherein the partitioning step includes:

scanning a portion of the document text stream;

comparing the scanned portion of the document text stream to partition entities in the partition list;

substituting a partition tag for portions of the document text stream which match a partition entity;

generating a text chunk list;

scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks.

3. The method of claim 1, wherein the selecting step includes:

selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

4. The method of claim 1, wherein:

a) the partitioning step includes:

a1) scanning a portion of the document text stream;

a2) comparing the scanned portion of the document text stream to partition entities in the partition list;

a3) substituting a partition tag for portions of the document text stream which match a partition entity;

a4) generating a text chunk list;

a5) scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

a6) revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks; and

b) the selecting step includes selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

5. The method of claim 1, wherein the selecting step includes:

excluding a chunk from being determined as a phrase if the chunk is a single word beginning with a lower case letter.

6. The method of claim 1, wherein the selecting step includes:

determining a chunk as being a phrase if the chunk includes a plurality of words each constituting lower case letters only if the chunk occurs at least twice in the document text stream.

7. The method of claim 1, wherein the selecting step includes:

determining a chunk as being a proper name if the chunk includes a plurality of words each having at least a first letter which is upper case.

8. The method of claim 1, wherein the selecting step includes:

mapping a sub-phrase to a phrase.

9. The method of claim 1, wherein the selecting step includes:

mapping single upper case words to their respective proper names.

10. The method of claim 1, wherein the selecting step includes:

detecting presence of acronyms;

incrementing a count of a proper name corresponding to the respective detected acronyms; and

copying the proper name and the acronym to an acronym list.

11. The method of claim 1, wherein the selecting step includes:

combining a phrase list of lower case words with a phrase list of proper names.

12. The method of claim 1, further comprising:

reducing the phrase list by consolidating phrases in the phrase list by using a synonym thesaurus.

13. The method of claim 1, further comprising:

adding phrases to the phrase list by combining phrases which are separated in the document text stream only by prepositions.

14. The method of claim 1, further comprising:

trimming the phrase list by eliminating phrases which occur in fewer than a threshold number of document text streams.

15. The method of claim 1, further comprising:

categorizing proper names in the proper name list into groups based on corresponding group lists.

16. An apparatus of processing a stream of document text to form a list of phrases that are indicative of conceptual content of the document, the phrases being used as index terms and search query terms in full text document searching performed after the phrase list is formed, the apparatus comprising:

means for partitioning the document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

means for selecting certain chunks as the phrases of the phrase list, based on frequencies of occurrence of the chunks within the stream of document text.

17. The apparatus of claim 16, wherein the partitioning means includes:

means for scanning a portion of the document text stream;

means for comparing the scanned portion of the document text stream to partition entities in the partition list;

means for substituting a partition tag for portions of the document text stream which match a partition entity;

means for generating a text chunk list;

means for scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

means for revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks.

18. The apparatus of claim 16, wherein the selecting means includes:

means for selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

19. The apparatus of claim 16, wherein:

a) the partitioning means includes:

a1) means for scanning a portion of the document text stream;

a2) means for comparing the scanned portion of the document text stream to partition entities in the partition list;

a3) means for substituting a partition tag for portions of the document text stream which match a partition entity;

a4) means for generating a text chunk list;

a5) means for scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

a6) means for revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks; and

b) the selecting means includes means for selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

20. A computer-readable memory which, when used in conjunction with a computer, can carry out a phrase recognition method to form a phrase list containing phrases that are indicative of conceptual content of a document, the phrases being used as index terms and search query terms in full-text document searching performed after the phrase list is formed, the computer-readable memory comprising:

computer-readable code for partitioning document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

computer-readable code for selecting certain chunks as the phrases of the phrase list based on frequencies of occurrence of the chunks within the stream of document text.

21. The computer-readable memory of claim 20, wherein the computer-readable code for partitioning includes:

computer-readable code for scanning a portion of the document text stream;

computer-readable code for comparing the scanned portion of the document text stream to partition entities in the partition list;

computer-readable code for substituting a partition tag for portions of the document text stream which match a partition entity;

computer-readable code for generating a text chunk list;

computer-readable code for scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

computer-readable code for revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks.

22. The computer-readable memory of claim 20, wherein the computer-readable code for selecting includes:

computer-readable code for selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

23. The computer-readable memory of claim 20, wherein:

a) the computer-readable code for partitioning includes:

a1) computer-readable code for scanning a portion of the document text stream;

a2) computer-readable code for comparing the scanned portion of the document text stream to partition entities in the partition list;

a3) computer-readable code for substituting a partition tag for portions of the document text stream which match a partition entity;

a4) computer-readable code for generating a text chunk list;

a5) computer-readable code for scanning the text chunk list to determine a frequency of each text chunk in the text chunk list; and

a6) computer-readable code for revising the text chunk list to include the respective frequencies of occurrence in association with the text chunks; and

b) the computer-readable code for selecting includes computer-readable code for selecting the certain chunks as the phrases of the phrase list based only on the frequencies of occurrence of the chunks within the stream of document text and on a quantity of words within the chunks.

24. A computer-implemented method of full-text, on-line searching, the method comprising:

a) receiving and executing a search query to display at least one current document;

b) receiving a command to search for documents having similar conceptual content to the current document;

c) executing a phrase recognition process to extract phrases allowing full text searches for documents having similar conceptual content to the current document, the phrase recognition process including the steps of:

c1) partitioning the document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

c2) selecting certain chunks as the phrases, based on frequencies of occurrence of the chunks within the stream of document text; and

d) automatically forming a second search query based at least on the phrases determined in the phrase recognition process so as to allow automated searching for documents having similar conceptual content to the current document.

25. The method of claim 24, further comprising:

validating phrases recognized by the phrase recognition process against phrases in a phrase dictionary before automatically forming the second search query.

26. The method of claim 24, further comprising:

displaying an error message if less than a threshold number of phrases are recognized for the current document.

27. A computer-implemented method of forming a phrase list containing phrases that are indicative of conceptual content of each of a plurality of documents, which phrases are used as index terms or in document search queries formed after the phrase list is formed, the method comprising:

a) selecting document text from the plurality of documents;

b) executing a phrase recognition process including the steps of:

b1) partitioning the document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

b2) selecting certain chunks as the phrases, based on frequencies of occurrence of the chunks within the stream of document text; and

c) forming the phrase list, wherein the phrase list includes:

1) phrases extracted by the phrase recognition process; and

2) respective frequencies of occurrence of the extracted phrases.

28. The method of **27**, further comprising:

forming a modified phrase list having only those phrases whose respective frequencies of occurrence are greater than a threshold number of occurrences.

29. The method of **27**, further comprising:

forming a phrase dictionary based on the phrase list formed in the forming step.

30. A computer-implemented method of forming phrase lists containing phrases that are indicative of conceptual content of documents, which phrases are used as index terms or in document search queries formed after the phrase list is formed, the method comprising

a) selecting document text from a sampling of documents from among a larger collection of documents; and

b) executing a phrase recognition process to extract phrases to form a phrase list for each document processed, the phrase recognition process including:

b1) partitioning the document text into plural chunks of document text, each chunk being separated by at least one partition entity from a partition list; and

b2) selecting certain chunks as the phrases of the phrase list based on frequencies of occurrence of the chunks within the stream of document text.

* * * * *

US005442778A

United States Patent [19]

Pedersen et al.

[11] Patent Number: 5,442,778

[45] Date of Patent: Aug. 15, 1995

[54] SCATTER-GATHER: A CLUSTER-BASED METHOD AND APPARATUS FOR BROWSING LARGE DOCUMENT COLLECTIONS

[75] Inventors: Jan. O. Pedersen, Palo Alto; David Karger, Stanford; Douglass R. Cutting, Menlo Park, all of Calif.; John W. Tukey, Princeton, N.J.

[73] Assignee: Xerox Corporation, Stamford, Conn.

[21] Appl. No.: 790,316

[22] Filed: Nov. 12, 1991

[51] Int. Cl.[6] G06F 17/30

[52] U.S. Cl. 395/600; 364/419.19; 364/419.13; 364/225.6; 364/DIG. 1

[58] Field of Search 395/600, 144; 382/39; 340/146.2; 364/419.19, 419.13

[56] References Cited

U.S. PATENT DOCUMENTS

3,568,155	3/1971	Abraham et al.	395/600
4,053,871	10/1977	Vidalin et al.	340/146.2
4,575,798	3/1986	Lindstrom et al.	395/600
4,774,655	9/1988	Kollin et al.	395/600
4,774,657	9/1988	Anderson et al.	395/600
4,956,774	9/1990	Shibamiya et al.	395/600
4,972,349	11/1990	Kleinberger	395/144
5,075,896	12/1991	Wilcox et al.	382/39
5,168,565	12/1992	Morita	395/600
5,257,365	10/1993	Powers et al.	395/600
5,278,980	1/1994	Pederson et al.	395/600

OTHER PUBLICATIONS

van Rijsbergen, Information Retrieval, second edition, 1979, pp. 1-65, 184-191.

Crouch, C. J., "A Cluster-based Approach to Thesaurus Construction", ACM SIGIR 11th International Conference on Research & Development in Information Retrieval, pp. 309-320, Jun. 15, 1988.

"Information Theater Could Overthrow Traditional Information Refiners"; D. Coursey, MIS Week Jun. 25, 1990.

"Information Theater Versus Information Refinery"; D. Cutting et al (Xerox). Mar. 27, 1990 Stanford University.

"Understanding Multi-Articled Documents" Tsujimoto et al.; Jun. 1990; Proc. 10th International Conference of Pattern Recognition.

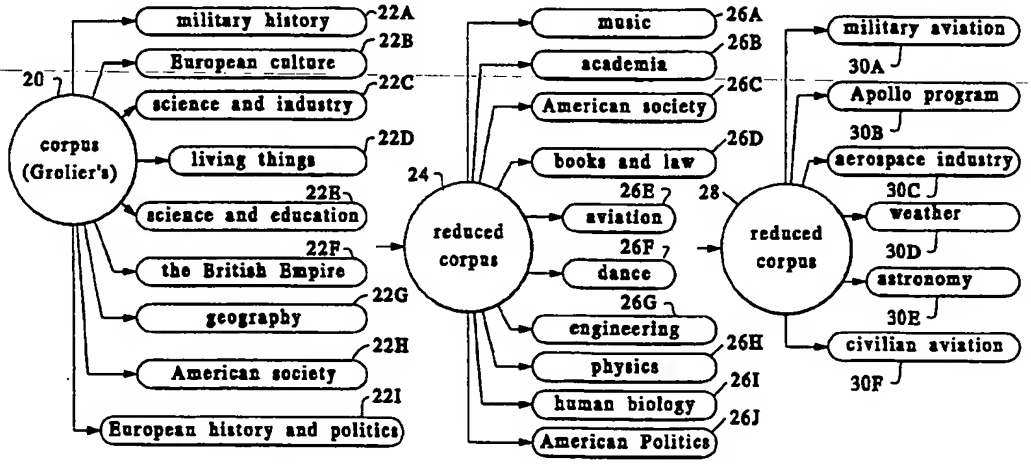
"Recent Trends in Hierarchic Document Clustering: A Critical Review"; P. Willett; Information Processing & Management, vol. 24, No. 5, pp. 577-597, 1988, Permagon Press, Inc.

Primary Examiner—Thomas G. Black
Assistant Examiner—Larry J. Ellcessor
Attorney, Agent, or Firm—Oliff & Berridge

[57] ABSTRACT

Scatter-Gather is a computer based document browsing method which operates in time proportional to a number of documents in a target corpus. The Scatter-Gather method includes: preparing an initial ordering of the corpus using, for example, an off-line computational method; determining a summary of the initial ordering of the corpus for interactive utility; and providing a further ordering of the corpus using, for example, an on-line non-deterministic method. The step of an off-line preparation of an initial ordering of a corpus is non-time-dependent, thus an accurate initial ordering is prepared. The step of determining a summary includes determining a summary for presentation to a user without scrolling on a CRT. The step of providing a further ordering includes truncated group average agglomerate clustering, merging disjointed document sets, center finding, assign-to-nearest and other refinement methods.

21 Claims, 5 Drawing Sheets



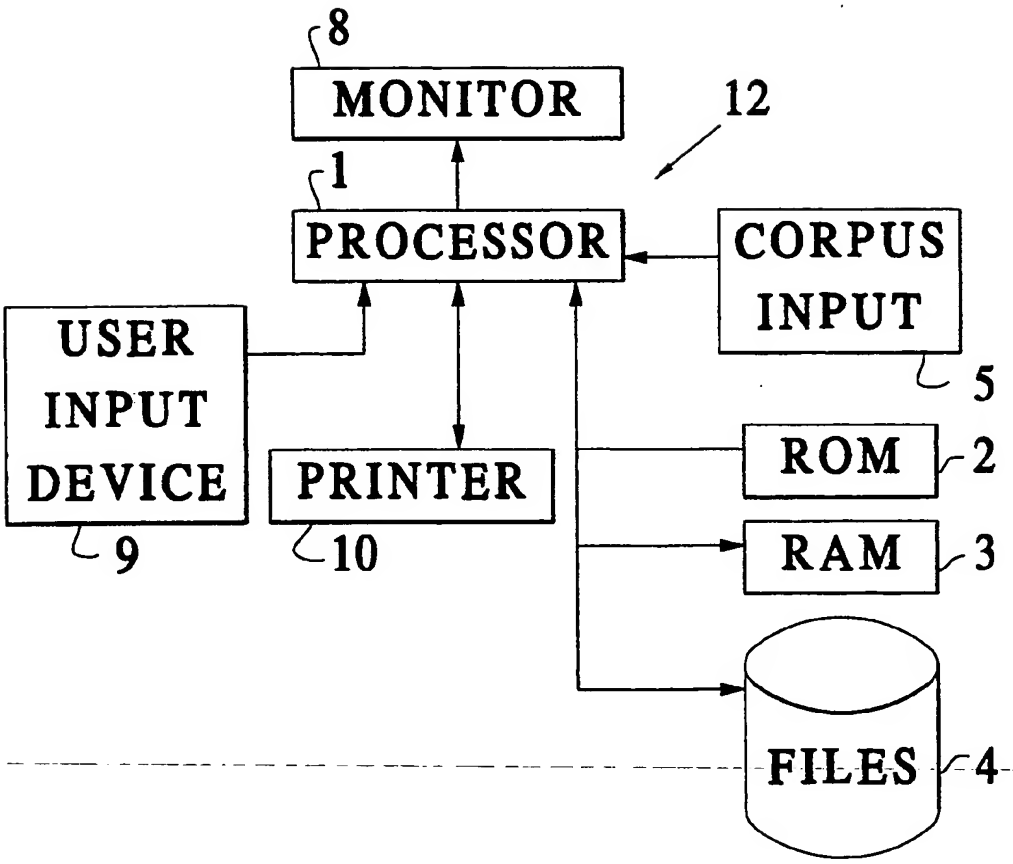

8
MONITOR
12
1
PROCESSOR
CORPUS INPUT
5
USER INPUT DEVICE
9
PRINTER
10
ROM
2
RAM
3
FILES
4

FIG. 1

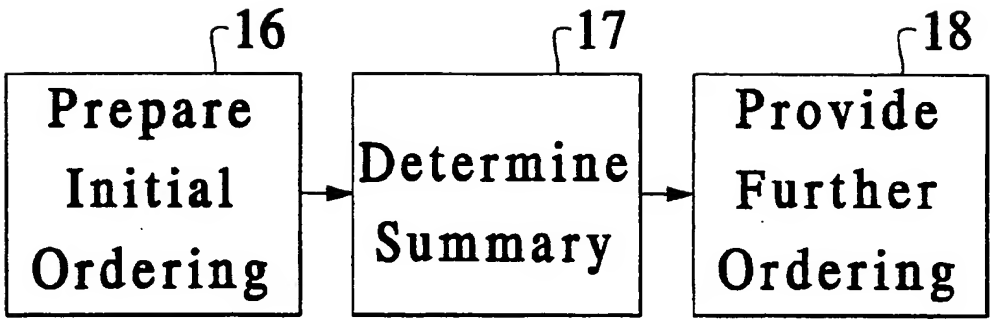

16
Prepare Initial Ordering
17
Determine Summary
18
Provide Further Ordering

Fig. 2

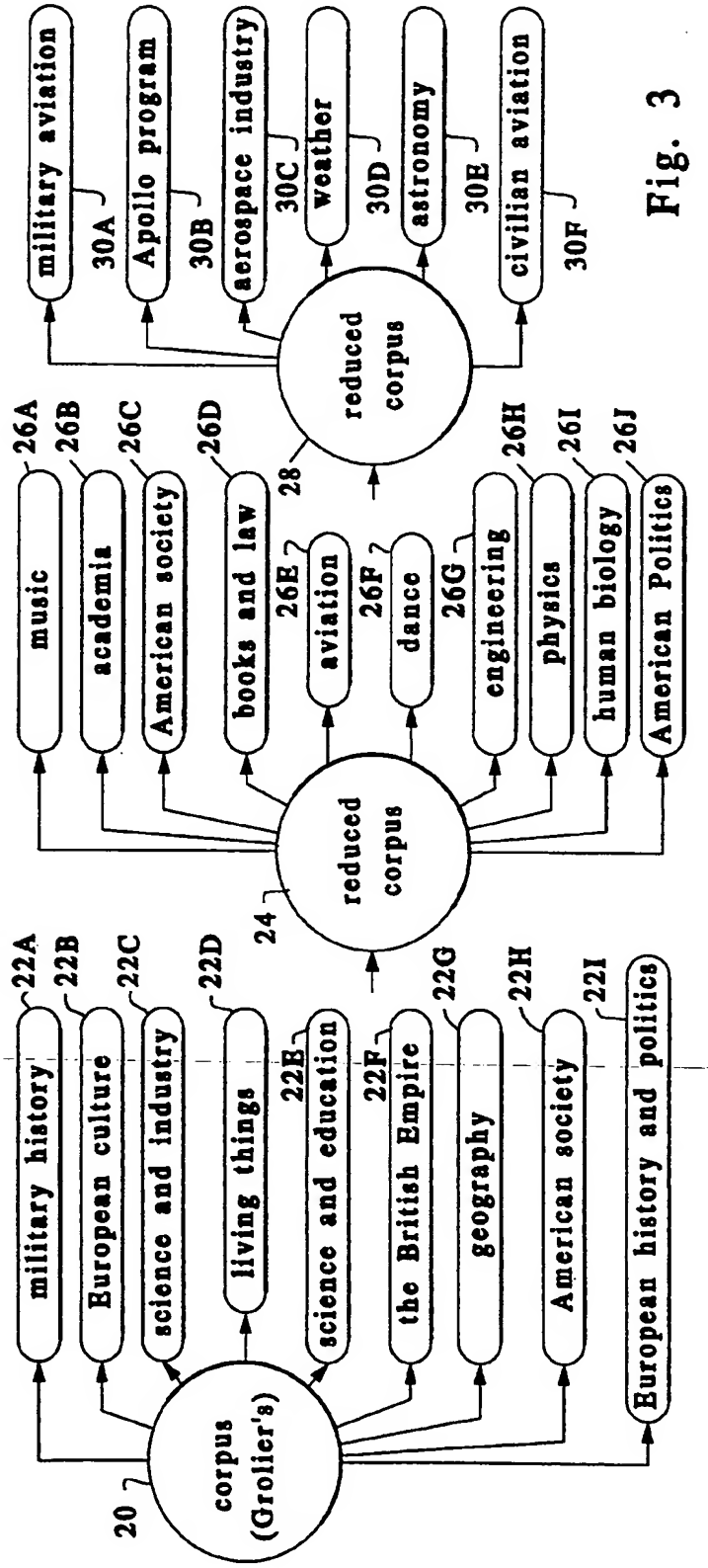

corpus (Grolier's)
20
military history
22A
European culture
22B
science and industry
22C
living things
22D
science and education
22E
the British Empire
22F
geography
22G
American society
22H
European history and politics
22I
reduced corpus
24
music
26A
academia
26B
American society
26C
books and law
26D
aviation
26E
dance
26F
engineering
26G
physics
26H
human biology
26I
American Politics
26J
reduced corpus
28
military aviation
30A
Apollo program
30B
aerospace industry
30C
weather
30D
astronomy
30E
civilian aviation
30F

Fig. 3

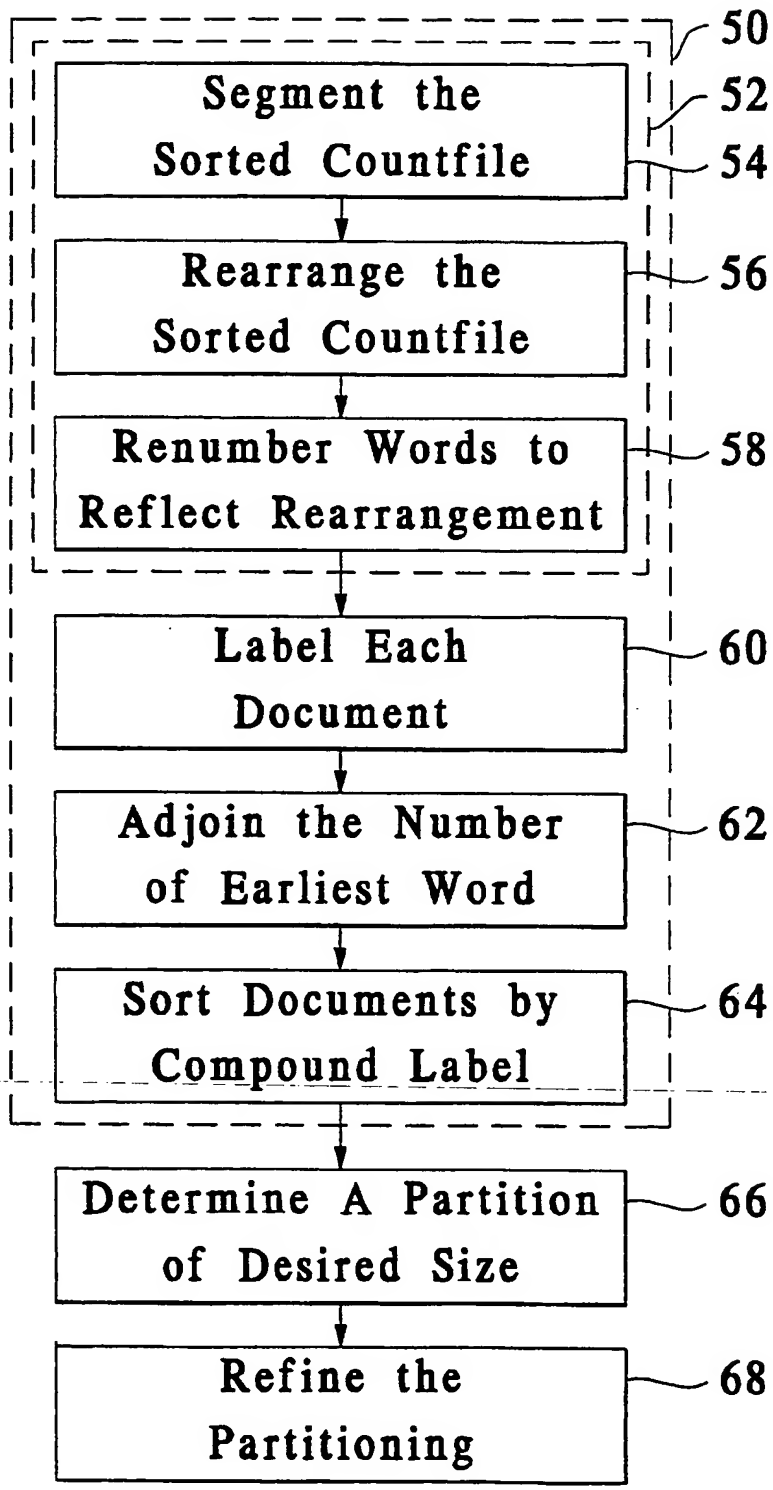


Fig. 4

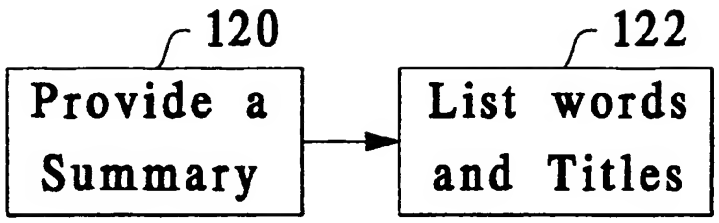
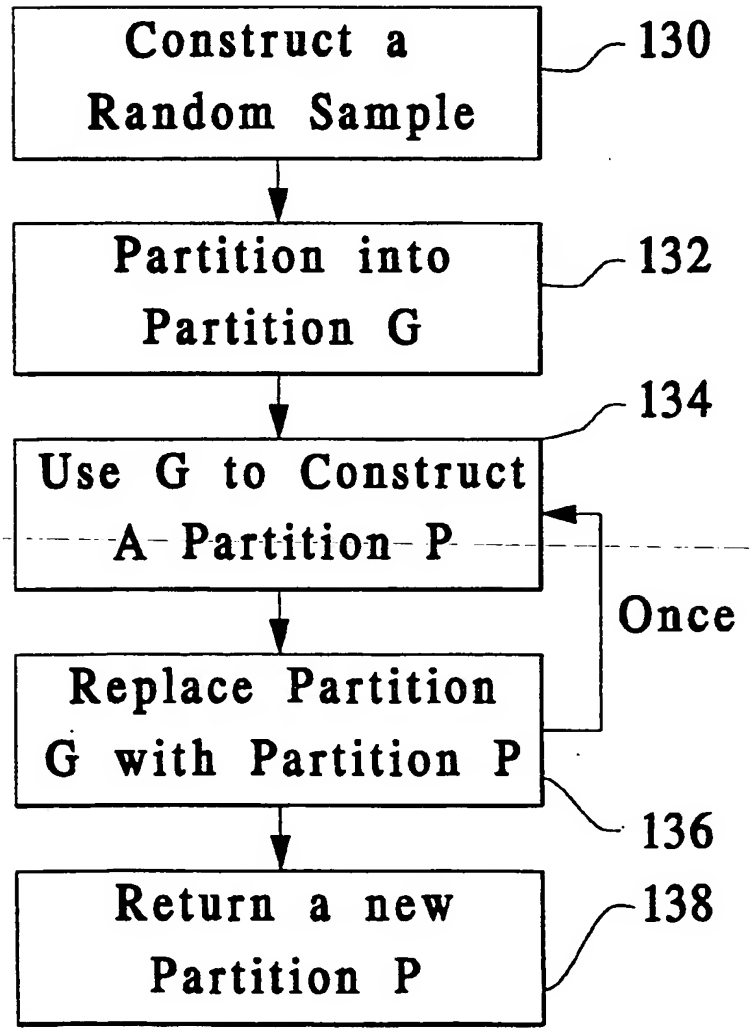

120
Provide a Summary
122
List words and Titles

Fig. 5

Fig. 6


Construct a Random Sample
130
Partition into Partition G
132
134
Use G to Construct A Partition P
Once
Replace Partition G with Partition P
136
Return a new Partition P
138

SCATTER-GATHER: A CLUSTER-BASED METHOD AND APPARATUS FOR BROWSING LARGE DOCUMENT COLLECTIONS

BACKGROUND OF THE INVENTION

The present invention relates to a document-clustering-based browsing procedure for a corpus of documents, which is applicable over all natural languages that contain a lexical analysis capability.

Document clustering has been extensively investigated as methodology for improving document search and retrieval. The general assumption is that mutually similar documents will tend to be relevant to the same queries and hence, automatic determination of groups of such documents can improve recall by effectively broadening a search request. Typically a fixed corpus of documents is clustered either into an exhaustive partition, disjoint or otherwise, or into an hierarchical tree structure. In the case of a partition, queries are matched against clusters, and the contents of some number of the best scoring clusters are returned as a result, possibly sorted by score. In the case of a hierarchy, queries are processed downward, always taking the highest scoring branch, until some stopping condition is achieved. The subtree at that point is then returned as a result.

Hybrid strategies are also available, which are essentially variations of near-neighbor search, where nearness is defined in terms of the pairwise document similarity measure used to generate the clustering. Indeed, cluster search techniques are typically compared to similarity search, a direct near-neighbor search, and are evaluated in terms of precision and recall. Various studies have indicated that cluster search strategies are not markedly superior to similarity search and, in some situations, can be inferior. It is therefore not surprising that cluster search, given its indifferent performance, and the high determinable cost of clustering large corpora, has not gained wide popularity.

Document clustering has also been studied as a method for accelerating similarity search, but the development of fast procedures for near-neighbor searching has decreased interest in that possibility.

In order to cluster documents, one must first establish a pairwise measure of document similarity and then define a method for using that measure to form sets of similar documents, or clusters. Numerous document similarity measures have been proposed, all of which consider the degree of word overlap between the two documents of interest, described as sets of words, often with frequency information. These sets are typically represented as sparse vectors of length equal to the number of unique words (or types) in the corpus. If a word occurs in a document, its location in this vector is occupied by some positive value (one if only presence/absence information is considered, or some function of its frequency within that document if frequency is considered). If a word does not occur in a document, its location in this vector is occupied by zero. A popular similarity measure, the cosine measure, determines the cosine of the angle between these two sparse vectors. If both document vectors are normalized to unit length, this is of course, simply the inner product of the two vectors. Other measures include the Dice and Jaccard coefficient, which are normalized word overlap counts. It has also been suggested that the choice of similarity measure has less qualitative impact on clustering results than the choice of clustering procedure.

A wide range of clustering procedures have been applied to documents including, most prominently, single-linkage hierarchical clustering. Hierarchical clustering procedures proceed by iteratively considering all pairs of similarities, and fusing the pair which exhibits the greatest similarity. They differ in the procedure used to determine similarity when one of the pairs is a document group, i.e., the product of a previous fusion. Single-linkage clustering defines the similarity as the maximum similarity between any two individuals, one from each half of the pair. Alternative methods consider the minimum similarity (complete linkage), the average similarity (group average linkage), as well as other aggregate measures. Although single-linkage clustering is known to have an unfortunate chaining behavior, typically forming elongated straggly clusters, it continues to be popular due to its simplicity, and the availability of an optimal space/time procedure for its determination.

Standard hierarchical document clustering techniques employ a document similarity measure and consider the similarities of all pairs of documents in a given corpus. Typically, the most similar pair is fused and the process iterated, after suitably extending the similarity measure to operate on agglomerations of documents as well as individual documents. The final output is a binary tree structure that records the nested sequence of pairwise joints. Traditionally, the resulting trees had been used to improve the efficiency of standard boolean or relevance searches by grouping together similar documents for rapid access. The resulting trees have also lead to the notion of cluster search in which a query is matched directly against nodes in the cluster tree and the best matching subtree is returned. Counting all pairs, the cost of constructing the cluster trees can be no less than proportional to N^2, where N is the number of documents in the corpus. Although clustering experiments have been conducted on corpora with documents numbering in the low tens of thousands, the intrinsic order of these clustering procedures works against the expectation that corpora will continue to increase in size. Similarly, although cluster searching has shown some promising results, the method tends to favor the most determinationally expensive similarity measures and seldom yields greatly increased performance over other standard methods.

Hierarchical methods are intrinsically quadratic in the number of documents to be clustered, because all pairs of similarities must be considered. This sharply limits their usefulness, even given procedures that attain this theoretical upper bound on performance. Partitional strategies (those that strive for a flat decomposition of the collection into sets of documents rather than a hierarchy of nested partitions) by contrast are typically rectangular in the size of the partition and the number of documents to be clustered. Generally, these procedures proceed by choosing in some manner, a number of seeds equal to the desired size (number of sets) of the final partition. Each document in the collection is then assigned to the closest seed. As a refinement the procedure can be iterated with, at each stage, a hopefully improved selection of cluster seeds. However, to be useful for cluster search the partition must be fairly fine, since it is desirable for each set to only contain a few documents. For example, a partition can be generated whose size is related to the number of unique words in the document collection. From this perspec-

tive, the potential determinable benefits of a partitional strategy are largely obviated by the large size (relative to the number of documents) of the required partition. For this reason partitional strategies have not been aggressively pursued by the information retrieval community.

The standard formulation of cluster search presumes a query, the user's expression of an information need. The task is then to search the collection of documents that match this need. However, it is not difficult to imagine a situation in which it is hard, if not impossible to formulate such a query. For example, the user may not be familiar with the vocabulary appropriate for describing a topic of interest, or may not wish to commit himself to a particular choice of words. Indeed, the user may not be looking for anything specific at all, but rather may wish to gain an appreciation for the general information content of the collection. It seems appropriate to describe this as browsing rather than search, since it is at one extreme of a spectrum of possible information access situations, ranging from requests for specific documents to broad, open-ended questions with a variety of possible answers. Standard information access techniques tend to emphasize search. This is especially clearly seen in cluster search where a technology capable of topic extraction, i.e., clustering, is submerged from view and used only as an assist for near-neighbor searching.

In proposing an alternative application for clustering in information access we take our inspiration from the access methods typically provided with a conventional text book. If one has a specific question in mind, and specific terms which define that question, one consults an index, which directs one to passages of interest, keyed by search words. However, if one is simply interested in gaining an overview, one can turn to the table of contents which lays out the logical structure of the text for perusal. The table of contents gives one a sense of the types of questions that might be answered if a more intensive examination of the text were attempted, and may also lead to specific sections of interest. One can easily alternate between browsing the table of contents, and searching the index.

By direct analogy, an information access system is proposed herein, which can have, for example, two components: a browsing tool which uses a cluster-based, dynamic table-of-contents metaphor for navigating a collection of documents; and one or more word-based, directed text search tools, such as similarity search, or the search technique described in U.S. patent application Ser. No. 07/745,794 to Jan O. Pedersen et al filed Aug. 16, 1991, and entitled An Iterative Technique For Phrase Query Formation and an Information Retrieval System Employing Same. The browsing tool describes groups of similar documents, one or more of which can be selected for further refinement. This selection/refinement process can be iterated until the user is directly viewing individual documents. Based on documents found in this process, or on terms used to describe document groups, the user may at any time switch to a more focused search method. In particular it is anticipated that the browsing tool will not necessarily be used to find particular documents, but may instead assist the user in formulating a search request, which will then be evaluated by some other means.

U.S. Pat. No. 4,956,774 to Shibamiya et al. discloses a method for selecting an access path in a relational database management system having at least one index. The first step is to select a number of most frequently occurring values of at least part of a key of the index. The number is greater than zero and less than the total number of such values. Statistics on the frequency of occurrence of the selected values are collected. An estimate of the time required to use the index as the access path is made, based at least in part on the index's most frequently occurring values statistics. The estimate is used as the basis at least in part for selecting an access path for the query. The database optimizer described is hierarchically organized in order of word frequency.

"Recent trends in hierarchic document clustering: A critical review" by Peter Willett, Information Processing of Management, Vol. 24, No. 5, pages 577-97 (1988—printed in Great Britain) describes the calculation of interdocument similarities and clustering methods that are appropriate for document clustering. The article further discusses procedures that can be used to allow the implementation of the aforementioned methods on databases of nontrivial size. The validation of document hierarchies is described using tests based on the theory of random graphs and on empirical characteristics of document collections that are to be clustered. A range of search strategies is available for retrieval from document hierarchies and the results are presented in a series of research projects that have used these strategies to search a cluster resulting from several different types of hierarchic agglomerative clustering methods. The article suggests that a complete linkage method is probably the most effective method in terms of retrieval performance; however, it is also difficult to implement in an efficient manner. Other applications of document clustering techniques are discussed briefly; experimental evidence suggests that nearest neighbor clusters, possibly represented as a network model, provide a reasonably efficient and effective means of including interdocument similarity information in document retrieval systems.

"Understanding Multi-Articled Documents" by Tsujimoto et al., presented in June 1990 in Atlantic City, N.J. at the 10th International Conference for Pattern Recognition, describes an attempt to build a method to understand document layouts without the assistance of character recognition results, i.e., the meaning of contents. It is shown that documents have an obvious hierarchical structure in their geometry which is represented by a tree. A small number of rules are introduced to transform the geometric structure into the logical structure which represents the semantics carried by the documents. A virtual field separator technique is employed to utilize information carried by a special constituent of documents such as field separators and frames, keeping the number of transformation rules small.

SUMMARY OF THE INVENTION

In the basic iteration of the proposed Scatter-Gather browsing tool, which can be defined as a method and apparatus for browsing large document collections, the user is presented with the descriptions (or summaries) of a fixed number of document groups. Based on these summaries, the user selects one or more of these groups for further study. These selected groups are gathered together to form a subcollection. The system then scatters (or reclusters) the new subcollection into the same fixed number of document groups, which are again presented to the user. With each successive iteration, since the number of documents decreases through selec-

tion while the number of groups remains fixed, the groups become smaller and therefore more detailed. Ultimately, when the groups become small enough, this process bottoms out by viewing individual documents. A history mechanism allows the user to backup from the results of an iteration, to either try a different selection or to backup further.

Scatter-Gather depends on the existence of two facilities. First, since clustering and reclustering is an essential part of the basic iteration, there is clearly a requirement for a procedure which can cluster a large number of documents within a time tolerable for user interaction (typically no more than about 60 seconds). For this strategy to scale with corpus size, some provision must also be made for dealing with corpora larger than can be clustered within this time allowance. Second, given a group of documents, some method for automatically summarizing the group must be specified. This cluster description must be sufficiently revealing for a user to gain a sense of the topic defined by the group, yet short enough for many descriptions to be appreciated simultaneously.

Three procedures are disclosed which can be used together to satisfy the above requirements, or used separately to enhance other corpus partitioning and/or searching techniques. A first procedure, Buckshot is a very fast partition-type clustering method suitable for an online reclustering preferable for Scatter-Gather. The Buckshot procedure employs three subprocedures. The first subprocedure is truncated group average agglomerative clustering which merges disjoint document sets, or groups, starting with individuals, until only a predetermined number of groups remain. At each step the two groups whose merger would produce the greatest average similarity are merged into a single new group. The second subprocedure determines a trimmed sum profile from selected documents closest to a document group centroid. The third subprocedure assigns individual documents to the closest center represented by one of the trimmed average profiles.

A second procedure, Fractionation, is another more precise, partition-type clustering method suitable for a static off-line partitioning of the entire corpus which can be presented first to the user. Fractionation can be thought of as a more careful version of Buckshot that trades speed for increased accuracy. In particular, the random sampling of a corpus of documents and the partitioning of the random sample by truncated group average agglomerative clustering (as provided in the Buckshot procedure), are replaced by a deterministic center finding subprocedure (for Fractionation). Fractionation also provides additional refinement iterations and procedures. Specifically, Fractionation provides initial partitioning of a corpus of documents using a deterministic center finding subprocedure. Then, by applying an Assign-to-Nearest subprocedure, a partition of a desired size is determined. Finally, the partitioning is refined through iterative refinement subprocedures.

A third procedure, Cluster Digest is a cluster summarization suitable as a cluster display method in Scatter-Gather. For Scatter-Gather, it is desirable for a summary to be constant in size, so a fixed number of summaries can be reliably fit onto a given display area. The Cluster Digest summary lists a fixed member of topical words plus the document titles of a few typical documents, where topical words are those that often occur in a cluster, and typical documents are those close to the cluster centroid (or trimmed centroid).

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described in detail with reference to the following drawings in which line reference numerals refer to like elements.

FIG. 1 is a block diagram of the hardware components used to practice the present invention.

FIG. 2 is a high level flowchart of a preferred embodiment of the Scatter-Gather document browsing method according to the present invention.

FIG. 3 is an illustrative diagram of a preferred embodiment of the Scatter-Gather document browsing method of the present invention being applied to a corpus of documents consisting of Grolier's Encyclopedia.

FIG. 4 is an intermediate level flowchart of a first embodiment of a Fractionation initial partitioning method of the present invention for preparing an initial ordering of a corpus of documents as used in the Scatter-Gather document browsing method.

FIG. 5 is a high level flowchart, according to the present invention, of an embodiment of a Cluster Digest method for determining a summary of an ordering of a corpus of documents in the Scatter-Gather document browsing method.

FIG. 6 is an intermediate level flowchart, according to the present invention, of a preferred embodiment of a Buckshot method for providing a further, preferably on-line, ordering of a corpus of documents in the Scatter-Gather browsing method.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A. Overview

The present invention can be implemented in a document corpus browsing system **12** as illustrated by block diagram in FIG. **1**. The system includes a central processing unit (microprocessor) **1** for receiving signals from, and outputting signals to various other components of the system, according to one or more programs run on microprocessor **1**. The system includes a read only memory (ROM) **2** for storing operating programs. A random access memory (RAM) **3** is provided for running the various operating programs, and additional files **4** could be provided for overflow and the storage of partitioned corpora used by the present invention in performing a search operation.

Prior to performing a browsing procedure, a document corpus is input from a corpus input **5**. The corpus is then partitioned by software controlling processor **1** according to the teachings of the present invention.

Monitor **8** is provided for displaying results of partitioning procedures, and for permitting the user to interface with the operating programs. A user input device **9** such as, for example, a mouse, a keyboard, a touch screen or combinations thereof is provided for input of commands by the operator. A printer **10** can also be provided so that hard copies of documents, as well as print-outs containing Cluster Digest summaries, can be printed.

The system **12** is based in a digital computer which can implement an off-line preparation of an initial ordering using, for example, the Fractionation method disclosed herein. The system **12** also determines a summary of the initial ordering of the corpus which can be displayed to user via monitor **8** or printer **10** for user

interaction. This summary can be determined by, for example, using the Cluster Digest method disclosed herein. After receiving appropriate instructions from a user via user input device 9, system 12 can perform a further ordering of the corpus using, for example, the on-line Buckshot method described herein.

An illustration of Scatter-Gather in operation is shown in FIG. 3. Here the text collection 20 is an on-line-version of Grolier's encyclopedia (roughly 64 Megabytes of ASCII text) with each of the twenty-seven thousand articles treated as a separate document. Suppose the user is interested in investigating the role of women in the exploration of space. Rather than attempting to express this information need as a formal query, the user instead selects a number of the top-level clusters referenced as 22A–I that, from their description, seem relevant to the topic of interest. In this case, the user selects the clusters 22A, 22C and 22H labelled "military history", "science and industry", and "American society" to form a reduced corpus 24 of the indicated subset of articles from Grolier's. (Note, the cluster labels are idealized in this illustration; the actual implementation produces cluster descriptions which are longer than would fit conveniently in this figure. However, the given labels are reasonable glosses of topics described by actual cluster summaries).

The reduced corpus is then reclustered on the fly to produce a new set of clusters 26A–J covering the reduced corpus 24. Since the reduced corpus contains a subset of the articles in Grolier's, these new clusters are at a finer level of detail than the top-level clusters. The user again selects clusters of interest. In this case, these include clusters 26E, 26G and 26H labeled "aviation", "engineering", and "physics". Again, a further reduced corpus 28 is formed and reclustered. The final set of clusters 30A–F includes clusters labeled for "military aviation", "Apollo program", "aerospace industry", "weather", "astronomy" and "civil aviation". At this stage the clusters are small enough for direct perusal via an exhaustive list of article titles. Assuming at least one article of interest is found, the user may find more articles of a similar nature in the same cluster, or may use a directed search method, based on the vocabulary of the found article or of the cluster description, to find additional articles.

The Scatter-Gather browsing technique can be demonstrated in more detail through the following sample session performed with a prototype implementation. The basic steps are illustrated in FIG. 2, with the output data (output via a monitor or printer) being illustrated below. The corpus is still Grolier's encyclopedia and the goal is to find documents discussing space explorers. To create the initial partition (Step 16), the Buckshot clustering method is first applied. Fractionation could also be used for this task, time permitting.

```
> (setq first (time (outline (all-docs tdb))))
cluster 27581 items
global cluster 525 items . . .
cluster sizes 38 29 45 42 41 46 26 124 115 19
assign to nearest . . . done
cluster sizes 1838 1987 3318 2844 2578 2026 1568 4511 5730 1181
assign to nearest . . . done
cluster sizes 1532 2206 3486 2459 2328 2059 1609 4174 6440 1288
Cluster 0: Size: 1532. Focus articles: American music, music, history of; Ita
    focus terms: music, opus, composer, century, musical, play, dance, style, i
Cluster 1: Size: 2206. Focus articles: Christianity; Germany, history o; church
    focus terms:church, king, roman, son, war, century, christian, emperor, jo
Cluster 2: Size: 3486. Focus articles: French literature; English literature;
    focus terms:novel, trans, play, eng, writer, life, poet, american, poem, s
Cluster 3: Size: 2459. Focus articles: education; university; graduate educate
    focus terms:university, study, school, state, american, theory, college, s
Cluster 4: Size: 2328. Focus articles: plant; fossil record; mammal; growth; o
    focus terms:water, year, cell, area, animal, body, disease, human, develop
Cluster 5: Size: 2059. Focus articles: bird; shorebirds; flower; viburnum; cac
    focus terms:specie, family, plant, flower, grow, genus, tree, leaf, white,
Cluster 6: Size: 1609. Focus articles: radio astronomy; space exploration; sta
    focus terms:right, star, earth, space, energy, surface, motion, line, fiel
Cluster 7: Size: 4174. Focus articles: Latin American art; art; American art a
    focus terms:city, century, art, plant, build, center, style, bc, museum, d
Cluster 8: Size: 6440. Focus articles: United States, his; United States; Euro
    focus terms; state, war, unite, year, government, world, american, area, ri
Cluster 9: Size: 1288. Focus articles: chemistry, history; organic chemistry;
    focus terms:chemical, element, compound, metal, numb, atom, water, process
real time 465953 msec
```

Each cluster is described with a two line display, via an application of Cluster Digest (Step 17). Clusters 6 (Astronomy), 8 (U.S. History) and 9 (Chemistry) are picked, as those which seem likely to contain articles of interest, Recluster (Step 18), and Display.

```
> time (setq second (outline first 6 8 9)))
cluster 9337 items
global cluster 305 items . . .
cluster sizes 23 11 19 31 18 57 38 48 21 39
assign to nearest . . . done
cluster sizes 706 298 679 630 709 1992 980 1611 1159 573
assign to nearest . . . done
cluster sizes 538 315 680 433 761 1888 1376 1566 1068 712
Cluster 0: Size: 538. Focus articles: Liberal parties; political parties; Lab
    focus terms:party, minister, government, prime, leader, war, state, politi
Cluster 1: Size: 315. Focus articles: star; astronomy and astr; extragalactic
    focus terms:star, sun, galaxy, year, earth, distance, light, astronomer, m
```

-continued

```
Cluster 2: Size: 680. Focus articles: television; glass; aluminum; sound reco
    focus terms:process, metal, material, tight, type, mineral, color, device,
Cluster 3: Size: 433. Focus articles: Laccadive Islands; French Southern an;
    focus terms:island, sq, area, population, south, west, state, coast, north
Cluster 4: Size: 761. Focus articles: organic chemistry; chemistry, history;
    focus terms:chemical, element, compound, numb, atom, acid, reaction, water
Cluster 5: Size: 1888. Focus articles: United States, his; Europe, history of;
    focus terms:war, state, world, unite, american, british, army, government,
Cluster 6: Size: 1376. Focus articles: president of the U; Democratic party; b
    focus terms:state, president, law, court, year, unite, right, american, wa
Cluster 7: Size: 1566. Focus articles: United States; North America; Australia
    focus terms:state, river, area, population, north, south, year, west, regi
Cluster 8: Size: 1068. Focus articles: space exploration; radio astronomy; ele
    focus terms;energy, space, light, earth, particle, field, theory, motion,
Cluster 9: Size: 712. Focus articles: corporation; monopoly and compe; govern
    focus terms:company, state, unite, product, billion, year, service, sale,
real time 186338 msec
```

Iteration occurs, this time selecting clusters 1 (Astronomy) and 8 (Space Exploration and Astronomy). Space exploration is well separated from Astronomy in cluster 7, thus Space Exploration is picked by the operator.

```
> (time (setq third (outline second 1 8)))
cluster 1383 items
global cluster 117 items . . .
cluster sizes 12 4 8 15 12 7 8 22 9 20
move to nearest . . . done
cluster sizes 172 63 70 236 76 131 75 198 124 238
move to nearest . . . done
cluster sizes 176 83 86 205 86 134 84 187 132 210
Cluster 0: Size: 176. Focus articles: thermodynamics; light; energy; optics;
    focus terms:energy, heat, light, temperature, gas, wave, motion, air, pres
Cluster 1: Size: 83. Focus articles: analytic geometry; line; geometry; coor
    focus terms:point, line, plane, angle, circle, geometry, coordinate, curve
Cluster 2: Size: 86. Focus articles: radio astronomy; observatory, astro; te
    focus terms:telescope, observatory, radio, instrument, astronomy, light, s
Cluster 3: Size: 205. Focus articles: solar system; Moon; planets; astronomy,
    focus terms:earth, sun, solar, planet, moon, satellite, orbit, surface, ye
Cluster 4: Size: 86. Focus articles: radio, magnetism; circuit, electric; ge
    focus terms:field, frequency, magnetic, electric, electrical, wave, circui
Cluster 5: Size: 134. Focus articles: nuclear physics; atomic nucleus; physic
    focus terms:particle, energy, electron, charge, nuclear, proton, radiation
Cluster 6: Size: 84. Focus articles: measurements; units, physical; electroma
    focus terms:unit, value, measure, numb, measurement, function, equal, obje
Cluster 7: Size: 187. Focus articles: space exploration; Space Shuttle; Soyuz
    focus terms:space, launch, fright, orbit, satellite, mission, rocket, eart
Cluster 8: Size 132. Focus articles: physics, history o; Einstein, Albert; g
    focus terms;theory, physic, light, physicist, motion, einstein, law, parti
Cluster 9: Size: 210. Focus articles: star; extragalactic syst; astronomy and
    focus terms:star, galaxy, light, year, sun, distance, mass, cluster, brigh
real time 37146 msec
```

```
> (time (setq fourth (outline third 7)))
cluster 187 items
global cluster 43 items . . .
cluster sizes 1 2 4 1 1 15 1 11 2 5
assign to nearest . . . done
cluster sizes 1 6 20 5 2 79 1 47 4 22
assign to nearest . . . done
cluster sizes 1 9 22 8 2 69 1 46 4 25
Cluster 0: Size: 1. Focus articles: Stealth
    focus terms:radar, bomber, aircraft, fly, stealth, shape, wing, replace, a
Cluster 1: Size: 9. Focus articles: Juno; von Braun, Wernher; Jupiter; soun
    focus terms:rocket,space, missile, research, jupiter, redstone, satellite
Cluster 2: Size: 22. Focus articles: rockets and missil; Atlas; Thor; Titan;
    focus terms:missile, rocket, launch, stage, kg, thrust, space, ballistic,
Cluster 3: Size: 8. Focus articles: helicopter; VTOL; flight; jet propulsio
    focus terms:flight, engine, air, aircraft, rotor, helicopter, lift, speed,
Cluster 4: Size: 2. Focus articles: STOL; C-5 Galaxy;
    focus terms:aircraft, wing, speed, lift, engine, air, takeoff, land, weigh
Cluster 5: Size: 69. Focus articles: space exploration Soyuz; Salyut; Volko
    focus terms:space, launch, soyuz, cosmonaut, soviet, flight, spacecraft, m
Cluster 6: Size: 1. Focus articles: railgun;
    focus terms:projectile, sec, accelerate, speed, space, test, launch, field
Cluster 7: Size: 46. Focus articles: Gordon, Richard F.; Stafford, Thomas P;
    focus terms:astronaut, apollo, pilot, space, lunar, mission, flight, moon,
Cluster 8: Size: 4. Focus articles: phobia; claustrophobia; agoraphobia; He
```

-continued

focus terms;space, fear, phobia, claustrophobia, canvas, person, agoraphob
Cluster 9: Size: 25. Focus articles: communications sat; GEOS; Vanguard; SYN
focus terms:satellite, launch, orbit, space, earth, kg, communication, pro
real time 18251 msec

Two relevant, yet distinct clusters emerge at this stage; namely 5 (Soviet Space Exploration) and 7 (U.S. Space Exploration). The contents of these clusters is exhaustively displayed as follows:

(print-titles (nth 5 fourth))

40	Zond
74	Zholobov, Vitaly
238	Yelliseyev, Aleksei Stanislavovich
239	Yegorov, Boris B.
921	weightlessness
1269	Vostok
1270	Voskhod
1286	Volynov, Boris Valentinovich
1306	Volkov, Vladislav Nikolayevich
1574	Venera
2345	tracking station
2522	Titov, Gherman S.
2881	Tereshkova, Valentina
3959	Sputnik
4120	Spacelab
4125	space station
4126	Space Shuttle
4127	space medicine
4128	space law
4129	space exploration
4131	Soyuz
4365	SNAP
4477	Skylab
4849	Shatalov, Vladimir A.
4943	Sevastianov, Vitaly I.
5465	Sarafanov, Gennady Vasilievich
5611	Salyut
5809	Ryumin, Valery
5893	Rukavishnikov, Nikolay
5928	Rozhdestvensky, Valery
6074	Romanenko, Yury Viktorovich
6457	Remek, Vladimir
6652	Ranger
7381	Popovich, Pavel Romanovich
8267	Patsayev, Viktor
9751	National Aeronautics and Space Administration
10915	Mercury program
11104	McNair, Ronald
11437	Mars
11729	Makarov, Oleg
12006	Lyakhov, Vladimir
12042	Lunokhod
12054	Luna (spacecraft)
12616	Leonov, Aleksei
12723	Lebedev, Valentin
12766	Lazarev, V. G.
13145	Kubasov, Valery N.
13243	Komarov, Vladimir
13296	Klimuk, P. I.
13427	Khrunov, Yevgeny
13513	Kennedy, Space Center
13626	Kapustin Yar (ka-poos-tin yahr)
14072	Jarvis, Gregory
14224	Ivanov, Georgy
14226	Ivanchenkov, Aleksandr
16208	Gubarev, Aleksei
16439	Grechko, Georgy Mikhailovich
16665	Gorbatko, Viktor V.
16796	Godard Space Flight Center
16864	Glazkov, Yury Nokolayevich
18383	Feoktistov, Konstantin P.
19449	Dzhanibekov, Vladimir
19906	Dobrovolsky, Georgy T.
20266	Demin, Lev Stepanovich
23340	Bykovsky, Valery
25920	astronautics
26021	Artyukhin, Y. P.
26313	Apollo-Soyuz Test Project
27103	Aksenov, Vladimir

-continued

(print title (nth 5 fourth))

177	Young, John W.
403	Worden, Alfred M.
753	White, Edward H., II
903	Weitz, Paul J.
3391	Surveyor
3910	Stafford, Thomas P.
4460	Slayton, D. K.
4805	Shepard, Alan B., Jr.
5173	Scott, David R.
5211	Schweickart, Russell L.
5289	Schirra, Walter M., Jr.
6047	Roosa, Stuart A.
7519	Pogue, William Reid
10526	Mitchell, Edgar D.
11139	McDivitt, James
11245	Mattingly, Thomas
12050	Lunar Rover
12051	Lunar Orbiter
12052	Lunar Excursion Module
12148	Lousma, Jack
13897	Johnson Space Center
14297	Irwin, James
16026	Haise, Fred W.
16299	Grissom, Virgil I.
16651	Gordon, Richard F., Jr.
16998	Gibson, Edward
17189	Gemini program
17282	Garriott, Owen
18697	Evans, Ronald
19236	Eisele, Donn F.
19579	Duke, Charles, Jr.
20916	Crippen, Robert
21243	Cooper, Leroy Gordon, Jr.
21348	Conrad, Pete
21593	Collins, Michael
22479	Chaffee, Roger
22523	Cernan, Eugene
22826	Carr, Gerald
22835	Carpenter, Scott
23924	Brand, Vance
25140	Bean, Alan
25921	astronaut
26102	Armstrong, Neil A.
26314	Apollo program
26567	Anders, William Alison
26998	Aldrin, Edwin E.

The existence of two sets of relevant documents has been discovered with relatively disjoint vocabularies. At this stage individual documents may be examined, or some directed search tool might be applied to this restricted corpus. This example illustrates that the steps of determining a summary (with Cluster Digest), and providing a further ordering (with, for example, Buckshot) can be performed multiple times.

B. Procedures

For each document, α, in a collection (or corpus), C, let a countfile $c(\alpha)$ be a set of words, with their frequencies, that occur in that document. Let V be a set of unique words occurring in C. Then $c(\alpha)$ can be represented as a vector of length $|V|$;

$$c(\alpha)_i = \begin{cases} 0 & \text{if } w_i \notin \alpha \\ f(w_i,\alpha) & \text{if } w_i \in \alpha \end{cases}$$

for $1 \leqq i \leqq |V|$, where w_i is the ith word in V and $f(w_i,\alpha)$ is the frequency of w_i in α.

To measure the similarity between pairs of documents, α and β, let the cosine be employed between monotone element-wise functions of $c(\alpha)$ and $c(\beta)$. In particular, let

$$s(\alpha,\beta) = \frac{<g(c(\alpha)),g(c(\beta))>}{\|g(c(\alpha))\| \, \|g(c(\beta))\|}$$

where g is a monotone damping function and "<>" denotes inner product. $g(x)=\sqrt{x}$ produces good results hence, $g(x)=\sqrt{x}$ is used in the current implementation. However, it is understood that other values of g(x) could be used.

It is useful to consider similarity to be a function of document profiles, where

$$p(\alpha) = \frac{g(c(\alpha))}{\|g(c(\alpha))\|},$$

in which case

$$s(\alpha,\beta) = <p(\alpha),p(\beta)> = \sum_{i=1}^{|v|} p(\alpha)_i p(\beta)_i.$$

Suppose Γ is a set of documents, or a document group. A profile can be associated with Γ by defining it to be a normalized sum profile of contained individuals. Let

$$\hat{p}(\Gamma) = \sum_{\alpha \in \Gamma} p(\alpha),$$

define an unnormalized sum profile, and

$$p(\Gamma) = \frac{\hat{p}(\Gamma)}{\|\hat{p}(\Gamma)\|},$$

define a normalized sum profile. Similarly, the cosine measure can be extended to Γ by employing this profile definition:

$$s(\Gamma,x) = <p(\Gamma),p(x)>.$$

B.1. Partitional Clustering

Partitional clustering presumes a parameter k which is the desired size of the resulting partition (number of subgroups). The general strategy is to: (1) find k centers, or seeds; (2) assign each document to one of these centers (for each α in C assign α to the nearest center); and (3) possibly refine the partition, either through iteration or some other procedure. The result is a set P of k disjoint document groups such that $\bigcup_{\epsilon P} = C$.

The Buckshot method uses random sampling followed by truncated agglomerative clustering to find the k centers. Its refinement procedure is simply an iteration of assigning each document to a k center where new centers are determined from a previous partition P.

The Fractionation method uses successive stages of truncated agglomerative clustering over fixed sized groups to find the k centers. Refinement is achieved through repeated application of procedures that attempt to split, join and clarify elements of the partition P.

The Buckshot method sacrifices some precision (in the sense of document misclassification) in favor of speed, while Fractionation attempts to find a very high precision partition through exhaustive refinement. Buckshot is appropriate for the on-the-fly online reclustering required by inner iterations of Scatter-Gather, while Fractionation can be used, for example, to establish the primary partitioning of the entire corpus, which is displayed in the first iteration of Scatter-Gather.

B.2. Buckshot

The Buckshot method employs three subprocedures. The first subprocedure, truncated group average agglomerate clustering, merges disjoint document sets, or groups, starting with individuals until only k groups remain. At each step the two groups whose merger would produce the least decrease in average similarity are merged into a single new group. The second subprocedure determines a trimmed sum profile from selected documents closest to a document group centroid. The third subprocedure assigns individual documents to the closest center represented by one of these trimmed sum profiles.

For truncated group average agglomeration, let Γ be a document group, then the average similarity between any two documents in Γ is defined to be:

$$S(\Gamma) = \frac{2}{|\Gamma|(|\Gamma|-1)} \sum_{\alpha \in \Gamma} \sum_{\beta \in \Gamma} s(\alpha,\beta).$$

Let G be a set of disjoint document groups. The basic iteration of group average agglomerative clustering finds the pair Γ' and Δ' such that:

$$S(\Gamma' \cup \Delta') = \max_{\Gamma \epsilon G} (\max_{\Delta \neq \Gamma} S(\Gamma \cup \Delta)).$$

A new, smaller, partition G' is then constructed by merging Γ' with Δ'.

$$G' = (G - \{\Gamma', \Delta'\}) \cup \{\Gamma' \cup \Delta'\}.$$

Initially, G is simply a set of singleton groups, one for each individual to be clustered. The iteration terminates (or truncates) when $|G'| = k$. Note that the output from this procedure is the final flat partition G', rather than a nested hierarchy of partitions, although the latter could be determined by recording each pairwise join as one level in a dendrogram.

If the cosine similarity measure is employed, the inner maximization can be significantly accelerated. Recall that $\hat{p}(\Gamma)$ is the unnormalized sum profile associated with Γ. Then the average pairwise similarity, $S(\Gamma)$, is

simply related to the inner product, $\langle \hat{p}(\Gamma),\hat{p}(\Gamma)\rangle$. That is, since:

$$\begin{aligned}\langle \hat{p}(\Gamma),\hat{p}(\Gamma)\rangle &= \sum_{\alpha\in\Gamma}\sum_{\beta\in\Gamma}\langle p(\alpha),p(\beta)\rangle \\ &= |\Gamma|(|\Gamma|-1)S(\Gamma)+\sum_{\alpha\in\Gamma}\langle p(\alpha),p(\alpha)\rangle \\ &= |\Gamma|(|\Gamma|-1)S(\Gamma)+\Sigma\langle p(\alpha),p(\alpha)\rangle\end{aligned}$$

$$S(\Gamma)=\frac{\langle \hat{p}(\Gamma),\hat{p}(\Gamma)\rangle-|\Gamma|}{|\Gamma|(|\Gamma|-1)}.$$

Similarly, for the union of two disjoint groups, $\Lambda=\Gamma\cup\Delta$

$$S(\Lambda)=\frac{\langle \hat{p}(\Lambda),\hat{p}(\Lambda)\rangle(|\Gamma|+|\Delta|)}{(|\Gamma|+|\Delta|)((|\Gamma|+|\Delta|)-1)}$$

where,

$$\langle \hat{p}(\Lambda),\hat{p}(\Lambda)\rangle=\langle \hat{p}(\Gamma),\hat{p}(\Gamma)\rangle+2\langle \hat{p}(\Gamma),\hat{p}(\Delta)\rangle+\langle \hat{p}(\Delta),\hat{p}(\Delta)\rangle$$

Therefore, if for every $\Gamma\in G$, $S(\Gamma)$ and $\hat{p}(\Gamma)$ are known, the pairwise merge that will produce the greatest average similarity can be cheaply determined by determining one inner product for each candidate pair. Further, suppose for every $\Gamma\in G$ the Δ were known such that:

$$S(\Gamma\cap\Delta)=\max_{\Delta\neq\Gamma}S(\Gamma\cup\Delta),$$

then finding the best pair would simply involve scanning the $|G|$ candidates. Updating these quantities with each iteration is straight forward, since only those involving Γ' and Δ' need be redetermined.

Using techniques such as those described above, it can be seen that the average time complexity for truncated group average agglomerative clustering is $O(N^2)$, i.e., proportional to N^2, where N is equal to the number of individuals to be clustered.

B.2.a. Trimmed Sum Profiles

For trimmed sum profiles, given a set of k document groups that are to be treated as k centers for the purpose of attracting other documents, it is necessary to define a centroid, or attractor, for each group Γ. One simple definition would just consider the normalized sum profile for each group, $p(\Gamma)$. However, better results are achieved by trimming out documents far from this centroid.

For every α in Γ let $r(\alpha,\Gamma)$ be the rank of $\langle p(\alpha),p(\Gamma)\rangle$ in the set $\{\langle p(\beta),p(\Gamma)\rangle:\beta\in\Gamma\}$. If $r(\alpha,\Gamma)=1$ then $\langle p(\alpha),p(\Gamma)\rangle$ is the largest similarity in the set. Then define:

$$\hat{p}_m(\Gamma)=\sum_{\alpha 3 r(\alpha,\Gamma)\leq m}p(\alpha).$$

and

$$p_m(\Gamma)=\hat{p}_m(\Gamma)/\|\hat{p}(\Gamma)\|.$$

This determination can be carried out in time proportional to $|\Gamma|\log|\Gamma|$.

Essentially, $p_m(\Gamma)$ is the normalized sum of the m nearest neighbors in Γ to $p(\Gamma)$. The trimming parameter m may be defined adaptively as some percentage of $|\Gamma|$, or may be fixed. For example, in one implementation m=20.

The trimming of far away documents in determining the centroid profile leads to better focussed centers, and hence to more accurate assignment of individual documents in Assign-to-Nearest.

B.2.b. Assign to Nearest

For assign-to-nearest, once k centers have been found, and suitable profiles defined for those centers, each document in C must be assigned to one of those centers based on some criterion. The simplest criterion assigns each document to the closest center.

Let G be a partition of k groups, and let Γ_i be the ith group in G. Let:

$$\Pi_i=\left\{\alpha\in C:\langle \alpha,p_m(\Gamma_i)\rangle=\max_{j=0}^{k}\langle \alpha,p_m(\Gamma_j)\rangle\right\}.$$

Ties can be broken by assigning α to the group with lowest index. $P=\{\Pi_i\}$, $0\leq i\leq k$ is then the desired assign-to-nearest partition.

P can be efficiently determined by constructing an inverted map for the k centers $p_m(\Gamma_i)$, and for each $\alpha\in C$ simultaneously determining the similarity to all the centers. In any case, the cost of this procedure is proportional to kN, where $N=|C|$.

B.2.c Buckshot Procedure

FIG. 6 is a flowchart of a preferred embodiment of a Buckshot method for providing a further ordering of a corpus of documents in the Scatter-Gather method as shown in step 18 of FIG. 2. The Buckshot fast clustering method takes as input a corpus, C, an integer parameter k, $0<k\leq|C|$, and produces a partition of C into k groups. Let $N=|C|$. The steps of the Buckshot method include:

1. Construct a random sample C' from C of size $\sqrt{kN}$, sampling without replacement (step 130).
2. Partition C' into k groups by truncated group average agglomerative clustering (step 132). Call this partition G.
3. Construct a partition P of C by assigning each individual α to one of the centers in G (step 134). This is accomplished by applying assign-to-nearest over the corpus C and the k centers G.
4. Replace G with P (step 136) and repeat steps 3 and 4 once.
5. Return the new corpus partition P (step 138).

Since random sampling is employed in the definition of C', the Buckshot method is not deterministic. That is, repeated calls to this procedure on the same corpus may well produce different partitions; although repeated calls generally produce qualitatively similar partitions. It is easy to see that, if the C is actually composed of k well separated clusters, then as N increases, one is essentially assured of sampling at least one candidate from each cluster. Hence, asymptotically Buckshot will reliably find these k clusters.

Step 1 could be replaced by a more careful deterministic procedure, but at additional cost. In the contemplated application, Scatter-Gather, it is more important that the partition P be determined at high speed than that the partitioning procedure be deterministic. In-

deed, the lack of determination might be interpreted as a feature, since the user then has the option of discarding an unrevealing partition in favor of a fresh reclustering.

Step 4 is a refinement procedure which iterates assign-to-nearest once. Further iterations are of course possible, but with diminishing return. Since minimizing determine time is a high priority, one iteration of assign-to-nearest is a good compromise between accuracy and speed.

Steps 2, 3 and 4 all have time complexity proportional to kN. Hence the overall complexity of Buckshot is O(kN).

B.3 Cluster Digest

FIG. 5 is a flowchart of a preferred embodiment of a Cluster Digest method for determining a summary of an initial ordering of a corpus of documents in the Scatter-Gather document partitioning method as shown in step 17 of FIG. 2. For Cluster Digest, given a partition P, it is useful for diagnostic and display purposes to have some procedure for summarizing the individual document groups, or clusters, contained in P. One simple summary is simply an exhaustive list of titles, one for each document in . However, such a summary grows linearly with group size and hence, is of limited use for large clusters. For Scatter-Gather, it is desirable for the summary to be constant size, so a fixed number of summaries can be reliably fit onto a given display area. Thus, in step 120, a summary of constant size is provided, and in step 122, a fixed number of "topical" words plus the document titles of a few "typical" documents are listed. Here "topical" words are those that occur often in the cluster and "typical" documents are those close to the cluster centroid (or trimmed centroid). This summary is referred to as a Cluster Digest.

Cluster Digest takes as input a document group , and two parameters, ω, which determines the number of topical words output and d, which determines the number of typical documents found. The list of topical words is generated by sorting $p(\Gamma)$ (or $p_m(\Gamma)$) by term weight and selecting the ω highest weighted terms. The list of typical documents is generated by selecting the d documents closest to $p(\Gamma)$,

$$\{\alpha \epsilon \Gamma : r(\alpha,\Gamma) \leq d\}.$$

In one example, $\omega=10$ and $d=3$.

B.4 Fractionation

B.4.a. Overview and Supporting Material

Fractionation involves the development of an efficient and accurate document partitioning procedure which provides a number of desirable properties. The Fractionation procedure partitions a corpus into a predetermined number of "document groups". It operates in time proportional to the number of documents in the target corpus, thus distinguishing from conventional hierarchical clustering techniques which are intrinsically quadratictime. Although non-hierarchical in nature by producing a partition of the corpus rather than a tree, the Fractionation method can be applied recursively to generate a hierarchical structure if desired. New documents can be incorporated in time proportional to the number of buckets in an existing partition, providing desirable utility for dynamically changing corpora. Also, the resulting partitions are qualitively interesting, that is, with few exceptions buckets can be interpreted as topic coherent. Finally, the similarity measure at the base of the procedure depends only on term frequency considerations, and hence should be applicable over all natural languages that contain a lexical analysis capability.

First, a document corpus is divided into a number of buckets, each of a given fixed size. This initial division can be done at random or by making use of document similarity based on word overlap to induce an ordering over the corpus that places together documents likely to be similar. These fixed size buckets are individually clustered using a standard agglomerative procedure. However, a stopping rule removes agglomerations from consideration once they achieve a given size, for example, 5. Agglomerative clustering terminates when the ratio of current output agglomerations to original inputs falls below a given threshold, the reduction factor currently being, for example, 0.25. The outputs are collected and rearranged into fewer new buckets of the same fixed size where groups of documents, rather than individuals, are counted as employed in the initial division, and the output collection and rearranging process is reiterated. This stage of the Fractionation procedure completes when the total number of outputs from a given iteration is close to the desired number of partitions.

Although the basic building block of the procedure is agglomerative clustering, it is always applied to groups of a given fixed size. Each iteration can be thought of as producing, in a bottom-up fashion, one level of an n-ary branching tree where the branching factor is the reciprocal of the given reduction factor. Hence, agglomerative clustering is applied as many times as there are internal nodes of a tree. Since this number is proportional to the number of terminal nodes, the cost of the procedure is proportional to the cost of each agglomerative procedure times the number of terminal nodes. Thus, if n is the given fixed size,

$$(n^2)(N/n)=nN.$$

Therefore, this stage of the Fractionation procedure is linear in N for a fixed desired number of partitions.

In the agglomerative process, not every pairwise distance between agglomerations is considered, instead, reasonably well behaved subcorpora are separately clustered with the expectation that the seeds of global clusters will be formed within each subcorpus. The scope of the determination expands as it proceeds upwards, so that strong trends may be reinforced and weak ones subsumed. The final result is a partition of the original corpus into buckets where the content of each bucket is presumed to reflect the global trend. These buckets are expected to be noisy, in the sense that they may overlap in content, represent excessive merges, or contain documents better placed elsewhere. To repair these deficiencies, the initial partitioning is refined in iterative fashion.

A number of different refinement procedures can be devised which would improve the initial partitioning. Refinement methods which do not increase the linear order of the overall procedure can be accomplished by analysis of nearest neighbor patterns between individual documents of a given partitioning. It is desirable for the refinement efforts to be cumulative in the sense that they may be applied sequentially and in combination. Refinement methods can be used to merge, split and disperse buckets. Furthermore, application of a prede-

termined number of refinement methods does not change the linear order of the overall document procedure.

In contrast to standard hierarchical techniques, Fractionation can be easily adapted to incrementally update and dynamically change the partition of a given corpus. One simple strategy is to add new documents to the closest matching bucket of an existing partition. Refinement processes would run at regular intervals which could introduce new buckets if necessary by splitting old ones or merging buckets as distinctions become less clear. Also, although the resulting partition is not a hierarchical structure, it can be made so if desired, by recursive application of Fractionation to the subcorpora defined by single buckets.

Some particularly distinctive characteristics of Fractionation, as compared to other document partitioning procedures, include:

(1) Intermediate score. Rather than using word occurrence frequencies as a basis for document similarity, an intermediate score between the document and presence or absence in representation of document profiles is used. In particular, good performance is provided using square roots of counts within documents when summing the square roots across agglomerations. This allows putting about five documents or agglomerations together in a single step.

(2) Stepwise assembly. Preliminary buckets greater than or equal to five documents are not combined with single documents. Rather, stages of combining documents by five are iterated until the desired number of final buckets has been reached.

(3) Recurrent trimming. Documents which are not included in the initial preliminary bucketing into groups of greater than or equal to five documents are excluded from further procedures until the final iteration. At that time, the documents are assigned to the closet final bucket.

(4) Repeated improvement. Improvements are implemented through procedures such as repeated alternations of split and merge operations.

(5) No reliance on natural boundaries. Documents are partitioned according to internal coherence and reasonable distinctiveness, rather than by any comparison with predetermined boundaries.

B.4.b. The Fractionation Procedure

The overall procedure for Fractionation when starting with a new corpus, falls naturally into three stages, shown in FIG. 4:

Preparing an initial ordering of the corpus (step 50);

Determining a partitioning of the desired size from the initial ordering (step 66); and

Improving the partitioning by refinement (step 68).

Linear time is sustained because quadratic-time operations are restricted to subcorpora whose size does not grow with the size of the corpus.

Simple agglomeration applied to a large corpus is quadratic in the number of individuals. If it were possible to produce a reasonable initial partitioning of some given size in time proportional to the number of individuals, the application of any fixed number of refinement processes would not increase the order of the overall procedure. Further, if not initially then through refinement, the final result can be a good (internally coherent) partitioning of the corpus.

Since the basic tool for agglomeration is group average agglomerative clustering, it must be applied in a way that does not consider all pairwise similarities, or even a number proportional to all pairwise similarities. For a constant n which is similar to the desired size of the final partitioning, a corpus of N can be divided into groups of size n. Truncated agglomerative clustering may be applied to each group in time proportional to n^2. Since there about N/n groups, the total time for this step is proportional to $(N/n)n^2=Nn$.

Simple agglomerative clustering may be elaborated by supplying alternative stopping rules. These come in two types: conditions which limit the growth of any one agglomeration and conditions which terminate the overall process before a single over-arching agglomeration is formed. A condition of the first type states that if an agglomeration grows to size k, then it is removed from consideration in any further agglomeration steps. A condition of the second type terminates the iteration if the total set of agglomerations currently under consideration represents no more than r% of the size of the original corpus. Let k be the "stop" parameter and r be the "reduction" parameter.

Suppose two constants b and k are chosen, and agglomerative clustering is applied with reduction 1/b. That is, agglomeration stops as soon as fewer than n/b objects remain. After one step, the N/b outputs can be treated as individuals and iterated. This will take nN/b time. Thus the total time involving agglomerative clustering will be

$$nN(1 + b + b^2 \ldots) = \frac{b}{b-1} nN$$

which is rectangular in n and N. This process will be described as a center finding procedure, since it has frequently been used with k=5 and r=0.25, and may be viewed as a bottom-up construction of a 1/r branching tree.

The refinement procedures, to be discussed herein as well as the assign-to-nearest procedure and the multiple use of agglomerative clustering are all rectangular in the size of the number of individuals and either the number of buckets in the partitioning, or the size of the initial groups, n, which was chosen to approximate the size of the final partitioning.

FIG. 4 is a flowchart of one preferred embodiment of the Fractionation Method described herein which can be employed as the step of initializing a partition of a corpus of documents or preparing an initial ordering 16 as shown in FIG. 2.

Fractionation can be thought of as a more careful version of Buckshot that trades speed for increased accuracy. In particular, steps 1 and 2 of Buckshot are replaced by a deterministic center finding procedure, and step 4 is elaborated with additional refinement iterations and procedures. Since the refinement procedures have the capability to merge, split and destroy clusters, Fractionation may not produce exactly k document groups. In other words, Fractionation is an adaptive clustering method.

The center finding procedure finds k centers by initially breaking C into N/m buckets of a fixed size m. The individuals in these buckets are then separately agglomerated into document groups such that the reduction in number (from individuals to groups in each bucket) is roughly ρ. These groups are now treated as if they were individuals, and the entire process is re-

peated. The iteration terminates when only k groups remain. The center finding procedure can be viewed as building a $1/\rho$ branching tree bottom up, where the leaves are individual documents, terminating when only k roots remain.

Suppose the individuals in C are enumerated, so that $C=\alpha_1,\alpha_2, \ldots ,\alpha_N$. This ordering could reflect an extrinsic ordering on C, but a better procedure sorts C based on a key which is the word index of the Kth most common word in each individual, where $1 \leqq K < |V|$. Typically K is a smaller number, such as three, which favors medium frequency terms. This procedure thus encourages that nearby individuals in the corpus ordering have at least one word in common.

Group selection that provides relatively weak similarity of documents grouped together or placed in adjacent groups, will enhance the quality of the step-by-step bucketing of these groups. Thus, it is worthwhile to do some structuring (initial ordering) of the corpus into groups of adjacent elements of size n. One example of such an initial ordering process (box 50, FIG. 4) is based upon word similarity, and is described as follows:

Initial ordering

Input C, a corpus

1. Sort words (stem types) by entries in corpus countfile, most frequent first (Step 52).
   - 1.1 Segment the sorted countfile (Step 54) according to frequency>c (segment c), c>frequency≧d (segment d), and frequency≦d (segment e).
   - 1.2 Rearrange countfile (Step 56) in order segment d, then segment c, then segment e.
   - 1.3 Renumber words, (Step 58) to reflect this reordering.
2. Label (Step 60) each document by the number of the earliest word in the sorted file which appears in it.
3. Adjoin (Step 62) to this number the number of the first word (in text order) in the document.
4. Sort documents (Step 64) by the compound label (earliest in countfile, earliest in document).

The various steps take times of the following orders: Step 1 takes $|V|\log|V|$; step 2 takes $|C|$ time (multiplied by the average length of a documents profile); step 3 takes $|C|$ time; step 4 takes $|C|\log|C|$ time. Step 1.1 takes $|V|$ time, all of 1.1 to 1.2 are bounded by the total vocabulary size, and grow only slowly and boundedly as $|C|$ grows.

The center finding procedure builds an initial partitioning of a corpus C by first applying the structuring process to construct a crude grouping into groups of size n and then applying agglomerative clustering as previously described.

The initial bucketing creates a partition $B=\{\Theta_1, \Theta_2, \ldots ,\Theta_{N/m}\}$:

$$\Theta_i=\{\alpha_{m(i-1)+1},\alpha_{m(i-1)+2}, \ldots ,\alpha_{mi}\}$$

The document groups Θ_i are then separately clustered using an agglomerative procedure, such as truncated group average agglomerative clustering, with $k=\rho m$, where ρ is the desired reduction factor. Note that each of these determinations occurs in m^2 time and hence, all N/m occur in Nm time. Each application of agglomerative clustering produces an associated partition $R_i=\{\Phi_{i,1},\Phi_{1,2}, \ldots ,\Phi_{i,\rho m}\}$. The union of the documents groups contained in these partitions are then treated as individuals for the next iteration. That is, define

$$C'=\{\Phi_{i,j}: 1 \leqq i \leqq N/m,\ 1 \leqq j \leqq \rho m\}.$$

C' inherits an enumeration order by defining

$$\Phi_{i,j}=\alpha'_{N/m(i-1)+j}$$

The process is then repeated with C' instead of C. That is, the ρN components of C' are broken into $\rho N/m$ buckets, which are further reduced to $\rho^2 N$ groups that separate agglomeration. The process terminates at iteration j if $\rho^j N<k$. At this point one final application of agglomerative clustering can reduce the remaining groups to a partition P of size k (Step 66), for instance with m=200 and $\rho=\frac{1}{2}$.

The Fractionation refinement procedure (Step 68) performs a fixed number of iterations of a cycle of three separate operators. The first operator, Split, generates a new partition by dividing each document group in an existing partition into two parts. The second operator, Join, merges document groups that are indistinguishable from the point of view of Cluster Digest. The third operator is some number of iterations of Assign-to-Nearest, followed by the elimination of groups smaller than some threshold.

For every document group Γ in a partition P, the Split operator divides Γ into two new groups. This can be accomplished by applying Buckshot clustering with $C=\Gamma$ and k=2. The resulting Buckshot partition G provides the two new groups.

Let $P=\{\Gamma_1,\Gamma_2, \ldots ,\Gamma_k\}$ and let $G_i=\{\Gamma_{i,1},\Gamma_{i,2}\}$ be a two element Buckshot partition of Γ_i. The new partition P' is simply the union of the G_i's:

$$P = \bigcup_{i=j}^{k} G_i$$

Note that $|P'|=2k$. Each application of Buckshot requires time proportional to $2|\Gamma_i|$. Hence, the overall determination can be performed in time proportional to 2N.

A modification of this procedure would only split groups that score poorly on some coherency criterion. One simple coherency criterion is simply the average similarity to the cluster centroid:

$$A(\Gamma) = \frac{1}{|\Gamma|} \sum_{\alpha \in \Gamma} s(\alpha,\Gamma).$$

Let $r(\Gamma_i,P)$ be the rank of $A(\Gamma_i$ in the set $\{A(\Gamma_1),A(\Gamma_2), \ldots ,A(\Gamma_k)\}$. This procedure would then only split groups such that $r(\Gamma,P)<pk$ for some p, $0<p<1$. This modification does not change the order of the procedure since the coherence criterion can be determined in time proportional to N.

The purpose of the Join refinement operator is to merge document groups in a partition P that would be difficult to distinguish if they were summarized by Cluster Digest. Since by definition any two elements of P are disjoint, they will never have "typical" documents in common. However, their lists of "topical" words may well overlap. Therefore the criterion of distinguishability between two groups Γ and Δ will be:

$$T(\Gamma,\Delta)=|t(\Gamma) \cap t(\Delta)|$$

where t(Γ) is the list of topical words for Γ generated by the Cluster Digest summary.

Let Γ be related to Δ if $T(\Gamma,\Delta)>p$, for some p, $0<p\leqq w$. Let $\mathfrak{R}$ be the transitive closure of this relation. That is, $\mathfrak{R}$ (Γ,Δ) if there exists a sequence of groups $(\Lambda_1,\Lambda_2, \ldots ,\Lambda_m)$ such that $T(\Gamma,\Lambda_1)>p, T(\Lambda_i,\Lambda_{i+1})>p$ for $1\leqq i<m$, and $T(\Lambda_m,\Delta)>p$. Since $\mathfrak{R}$ is an equivalence relation, a new partition P' is generated by $\mathfrak{R}$. This partition is returned as the result of Join.

The determination of the transitive closure requires time proportional to k^2; hence the time complexity of Join is $O(k^2)$.

In one illustrative implementation w=10 and p=6.

The inputs of refinement are an existing partition P, a parameter I which determines the number of Split, Join, Assign-to-Nearest cycles executed, a parameter M which determines the number of Assign-to-Nearest iterations in each cycle, and a threshold T which is the minimum allowable document group size. A typical sequence of refinement steps follows:

Do I times
1 Split P to form P'
2 Join P' to form P''
3 Do M times
3.1 Apply Assign-to-Nearest to P'' to form G.
3.2 Eliminate Γ in G if $|\Gamma|<T$ to form G'
3.3 Let P''=G'
4 Return P''

The Elimination operator of Step 3.2 is implemented by supplying Assigning-to-Nearest with C=Γ and merging the resulting partition into G.

Steps 1, and 3.1 require time proportional to N, Step 2 requires time proportional to k^2, and step 3.2 requires time proportional to the number of individuals assigned to nearest, which is always less than N. Therefore, assuming $k^2<N$, which is typically the case, the overall time complexity of refinement is O(N).

In the illustrative implementation I=3, M=3 and T=10.

Fractionation takes as input a corpus C, and a parameter k, the desired number of clusters.

C. Summary

The Fractionation method as summarized includes the following steps:

1 Apply center finding to Construct an initial partition P
2 Apply Assign-to-Nearest with G=P to form P'
3 Apply Refinement to form P''
4 Return P'' as the final partition of C Note that, due to refinement, the size of the returned partition may not be equal to k.

In contrast to Buckshot, Fractionation is a deterministic procedure which returns the same partition if repeatedly called with the same values for C and k.

Step 1 is O(mN); step 2 O(kn); and step 3 O(N). Therefore assuming k<m, the overall time complexity of Fractionation is O(mN).

Scatter-Gather is an interactive browsing tool, and hence is better thought of as a collection of operators, rather than as a method that determines a particular quantity. It is presumed that a fixed corpus of interest, C, has been identified, and that the user is seated at a display (monitor 8) from which commands can be issued (via input device 9) to perform the next step. Since C is fixed, it is presumed that an initial partition can be determined off-line by application of the Fractionation clustering method. Fractionation is employed rather than Buckshot, since time is less of an issue for an off-line determination; hence time can be traded for accuracy. However, as noted in the earlier example, the initial ordering could be performed by Buckshot.

At start, this initial partition is presented to the user (via monitor 8 or printer 10) by applying Cluster Digest to each of the k document groups in order to determine a partition of a desired size as shown in step 17 of FIG. 2. Since Cluster Digest determines only two quantities of interest, a short list of typical document titles, and a short list of topical words, the presentation of each document group need not occupy more than a few lines of the display. In particular, it should be possible to present the entire partition in such a way that scrolling is not necessary to view all the summaries.

The operators available to the user are Pick, Recluster, Display and Backup. Given an existing partition, the Pick operation allows the user to select one or more document groups to form a new, reduced, subcorpus. The Recluster operator applies Buckshot to the current reduced subcorpus, defined by the last application of Pick. The Display operator presents the current partition by repeated application of Cluster Digest. The Backup operator replaces the current subcorpus with the subcorpus defined by the second most recent application of Pick.

A typical session iteratively applies a cycle of Pick, Recluster, and Display until the user comes across document groups of interest small enough for exhaustive display, or switches to the use of a directed search tool based on a term uncovered in the browsing process. A false Pick step can be undone by application of Backup.

While the present invention has been described with reference to particular preferred embodiments, the invention is not limited to the specific examples given and other embodiments and modifications can be made by those skilled in the art without departing from the spirit and scope of the invention.

What is claimed is:

1. A document browsing method in a digital computer for a corpus of documents, comprising the steps of:

preparing an initial ordering of the corpus into a first plurality of clusters by using a first method that automatically performs the initial ordering without external inputs based on contents of the documents using the digital computer;

determining a summary for each cluster of the first plurality of clusters prepared by said initial ordering of the corpus;

selecting by a user at least one cluster of the first plurality of clusters based on the summary of each cluster; and

automatically providing a further ordering of the user selected at least one cluster into a second plurality of clusters by automatically analyzing contents of documents of the selected at least one cluster using a second method comprising the steps of:

grouping together all of the documents from the selected at least one cluster based on the content of each document, and then

assigning each of the documents to one cluster of the second plurality of clusters.

2. The method of claim 1, wherein the preparing step includes a Fractionation method for partitioning the corpus of documents, said Fractionation method comprising the steps of:

preparing an ordering of the corpus;
determining a partitioning of a desired size from the ordering; and
refining the partitioning.

3. The method of claim 2, wherein the preparing an ordering step includes:
sorting words in order of frequency, most frequent word first, by entry into a corpus countfile;
labeling each document by a number of an earliest word in a sorted corpus countfile;
adjoining the number of an earliest word in a sorted corpus countfile to a number of a first text-ordered word in the document to form a compound label; and
sorting documents by the compound label.

4. The method of claim 3, wherein the sorting words step further comprises:
segmenting the sorted corpus countfile according to frequency into a number of segments;
rearranging the sorted corpus countfile according to segments; and
renumbering words to reflect the rearranging.

5. The method of claim 2, wherein the determining a partition step comprises truncated group averaging agglomerative clustering which includes limiting a growth of an agglomeration by terminating a group averaging agglomerative clustering before a single over-arching agglomeration is formed.

6. The method of claim 2, wherein the refining step includes refining with a assign-to-nearest method for assigning a document to a nearest bucket.

7. The method of claim 2, wherein the refining step includes merging similar buckets.

8. The method of claim 2, wherein the refining step includes splitting non-similar buckets.

9. The method of claim 2, wherein the refining step includes detecting at least one of weak similarity and small buckets and incoherent buckets by applying size and average similarity thresholds.

10. The document partitioning method of claim 1, wherein the determining a summary step includes determining a summary using a Cluster Digest method, said Cluster Digest method comprising the steps of:
providing a summary of constant size for each cluster; and
listing a fixed number of topical words plus document titles of a few typical documents within each cluster, wherein the topical words are words that occur often in the cluster and typical documents are documents close to a cluster centroid.

11. The document partitioning method of claim 10, wherein the providing a further ordering step includes providing a further ordering using a Buckshot method, said Buckshot method comprising the steps of:
constructing a random sample from the corpus of documents of size $\sqrt{kN}$ where k is an integer number of desired clusters and N is a number of documents in the corpus of documents;
partitioning into a partition G a random sample into k groups using truncated group average agglomerative clustering;
constructing a partition P of the corpus of documents by assigning each document to a k center in partition G and applying an assign-to-nearest procedure over the corpus and the k centers in partition G;
replacing partition G with partition P and repeating the step of constructing a partition; and
returning a new partition P.

12. The document partitioning method of claim 1, wherein the providing a further ordering step includes providing a further ordering step using a Buckshot method, said Buckshot method comprising the steps of:
constructing a random sample from the corpus of documents of size $\sqrt{kN}$ where k is an integer k number of desired clusters and N is a number of documents in the corpus of documents;
partitioning into a partition G a random sample into k groups using truncated group average agglomerative clustering;
constructing a partition P of the corpus of documents by assigning each document to a k center in partition G and applying assign-to-nearest over the corpus and the k centers in partition G;
replacing partition G with partition P and repeating the step of constructing a partition; and
returning a new partition P.

13. The document browsing method of claim 1, wherein the first method for preparing an initial ordering of the corpus is the same as the second method for providing a further ordering of a portion of the corpus.

14. A document browsing system for use with a corpus of documents in a digital computer, the document browsing system comprising:
preparing means for preparing without external inputs an initial ordering of the corpus into a first plurality of document clusters using the digital computer;
determining means for determining a summary for each cluster of the first plurality of document clusters prepared by said preparing means;
selecting means for a user to select at least one of the first plurality of document clusters; and
ordering means for automatically ordering the selected at least one of the first plurality of document clusters into a second plurality of clusters by
analyzing contents of documents of the selected at least one of the first plurality of document clusters,
grouping together all of the documents from the selected at least one of the first plurality of document clusters based on the contents of the documents of the selected at least one of the first plurality of document clusters, and then
assigning each of the documents to one cluster of the second plurality of clusters.

15. A document partitioning Fractionation method in a digital computer for non-hierarchical, linear-time partitioning of a corpus of documents, said Fractionation method comprising the steps of:
preparing an ordering of the corpus by
sorting words in order of frequency, most frequent word first, by entry into a corpus countfile,
labeling each document by a number of an earliest word in a sorted corpus countfile,
adjoining the number of an earliest word in a sorted corpus countfile to a number of a first text-ordered word in the document to form a compound label, and
sorting documents by the compound label;
determining a partitioning of a desired size from the ordering to form a set of buckets, each document of the corpus of documents assigned to only one bucket of the set of buckets; and
refining the partitioning by a predetermined number of iterations of

creating a the set of modified buckets from the set of buckets based on contents and size of each bucket

reassigning each document of the corpus of documents to the set of modified buckets.

16. The Fractionation method of claim **14**, wherein the sorting words step further comprises:

segmenting the sorted corpus countfile according to frequency into a number of segments;

rearranging the sorted corpus countfile according to segments; and

renumbering words to reflect the rearranging.

17. The Fractionation method of claim **14**, wherein the determining step comprises truncated group averaging agglomerative clustering which includes limiting a growth of an agglomeration by terminating a group averaging agglomerative clustering before a single over-arching agglomeration is formed.

18. The Fractionation method of claim **14**, wherein the refining step includes refining with a assign-to-nearest method for assigning a document to a nearest bucket.

19. The Fractionation method of claim **14**, wherein the refining step includes merging similar buckets.

20. The Fractionation method of claim **14**, wherein the refining step includes splitting non-similar buckets.

21. The method of claim **14**, wherein the refining step includes detecting at least one of weak similarity and small buckets and incoherent buckets by applying size and average similarity thresholds.

* * * * *

US005404514A

United States Patent [19]

Kageneck et al.

[11] Patent Number: 5,404,514

[45] Date of Patent: Apr. 4, 1995

[54] **METHOD OF INDEXING AND RETRIEVAL OF ELECTRONICALLY-STORED DOCUMENTS**

[76] Inventors: **Karl-Erbo G. Kageneck**, Jameson Hill Rd., Clinton Corners, N.Y. 12514; **Ted Young**, 300 Mercer St., Apt. 15A, New York, N.Y. 10003

[21] Appl. No.: **121,370**

[22] Filed: **Sep. 13, 1993**

Related U.S. Application Data

[63] Continuation of Ser. No. 998,023, Dec. 29, 1992, abandoned, which is a continuation-in-part of Ser. No. 456,558, Dec. 26, 1989, abandoned.

[51] **Int. Cl.**[6] **G06F 15/403**

[52] **U.S. Cl.** **395/600**; 364/419.19; 364/DIG. 2; 364/963; 364/963.4

[58] **Field of Search** 395/600; 364/419.19

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,495,566	1/1985	Dickinson et al.	395/600
4,674,066	6/1987	Kucera	395/600
5,062,074	10/1991	Kleinberger	395/600
5,168,565	1/1993	Morita	395/600
5,263,159	11/1993	Mitsui	395/600
5,301,109	4/1994	Landaver et al.	364/419.19
5,307,266	4/1994	Hayashi et al.	364/419.07

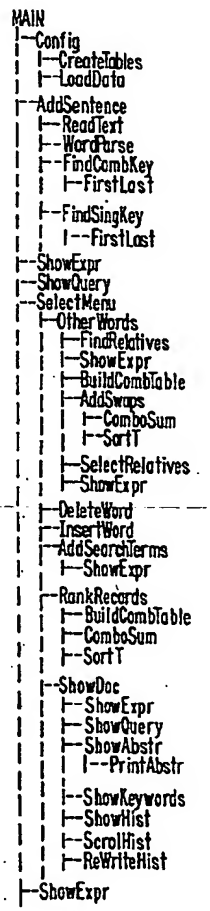
Primary Examiner—Paul V. Kulik

Attorney, Agent, or Firm—Cohen, Pontani, Lieberman, Pavane

[57] **ABSTRACT**

A document indexing and retrieval system and method which assigns weights to the key words and assigns a relative value to pairs of key words (i.e. defines a relative relation on K×K) based on their frequency of occurrence and co-occurrence in the document data base. In response to a query both the weights and this relative relation are used to suggest additional and/or alternative key words which are very likely to find relevant documents. Documents are then ranked by number of hits adjusted for the weights of hit words and their relative values.

6 Claims, 15 Drawing Sheets



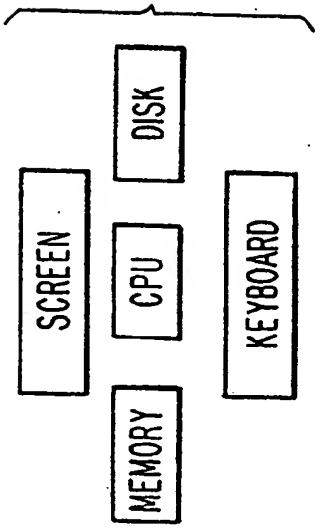

SCREEN
CPU
DISK
MEMORY
KEYBOARD

FIG. 1

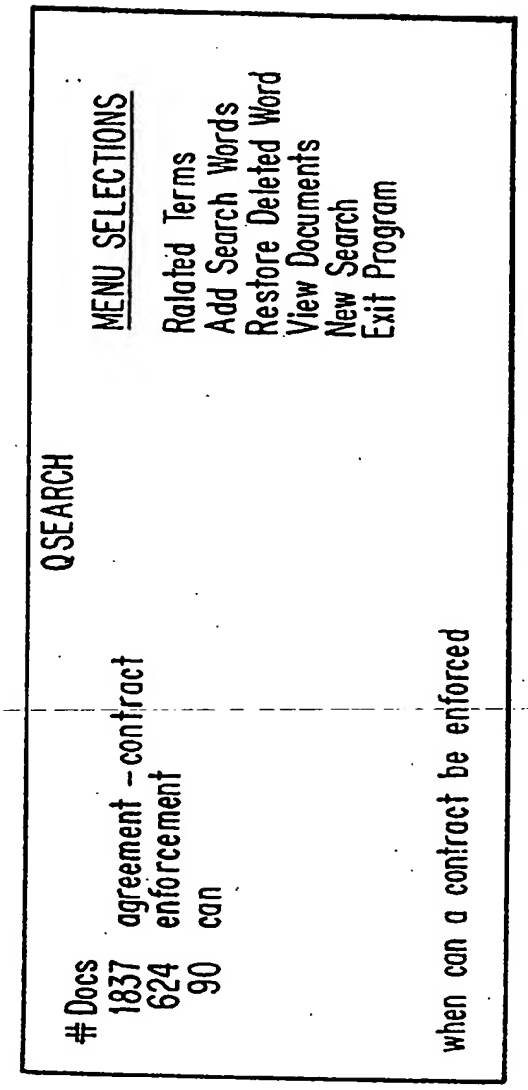

QSEARCH
MENU SELECTIONS
Related Terms
Add Search Words
Restore Deleted Word
View Documents
New Search
Exit Program
Docs
1837 agreement - contract
624 enforcement
90 can
when can a contract be enforced

FIG. 2

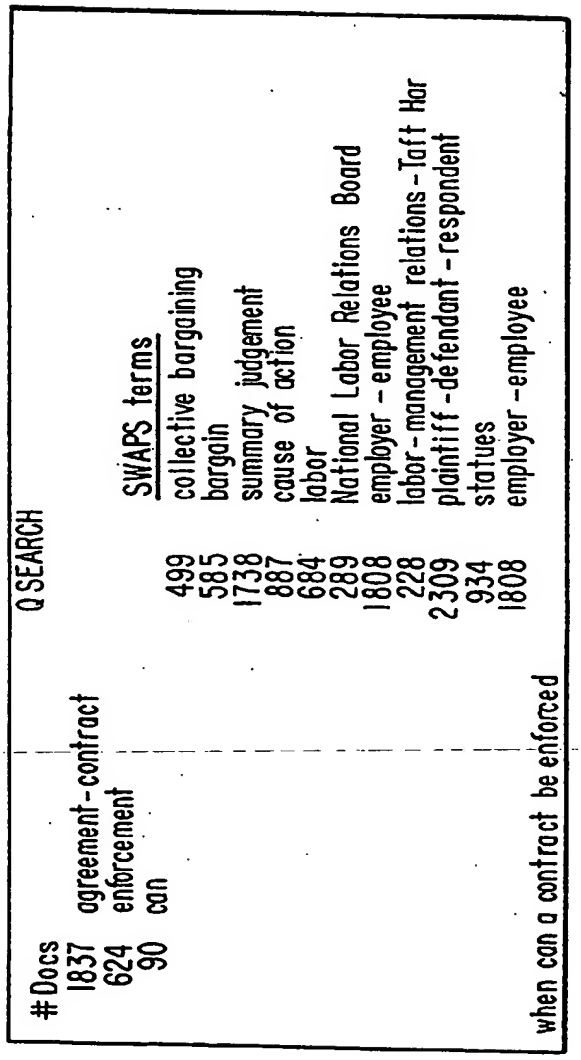

QSEARCH
#Docs
1837 agreement-contract
624 enforcement
90 can
SWAPS terms
499 collective bargaining
585 bargain
1738 summary judgement
887 cause of action
684 labor
289 National Labor Relations Board
1808 employer - employee
228 labor-management relations-Taft Har
2309 plaintiff-defendant-respondent
934 statues
1808 employer-employee
when can a contract be enforced

FIG. 3

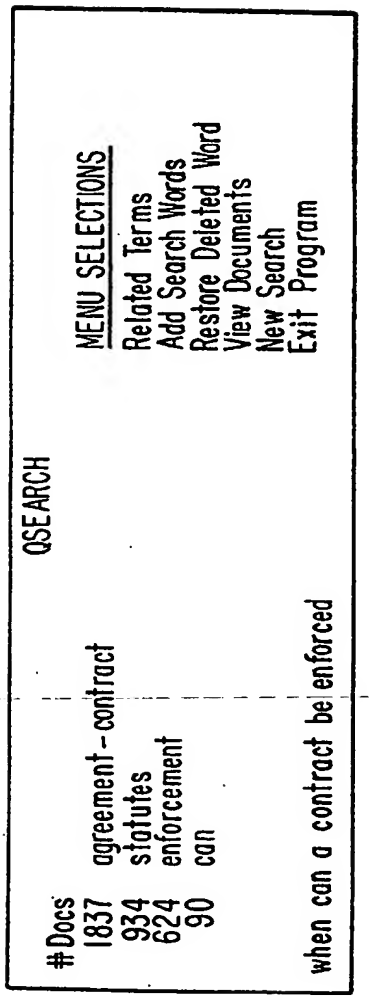

QSEARCH
#Docs
1837 agreement-contract
934 statutes
624 enforcement
90 can
MENU SELECTIONS
Related Terms
Add Search Words
Restore Deleted Word
View Documents
New Search
Exit Program
when can a contract be enforced

FIG. 4

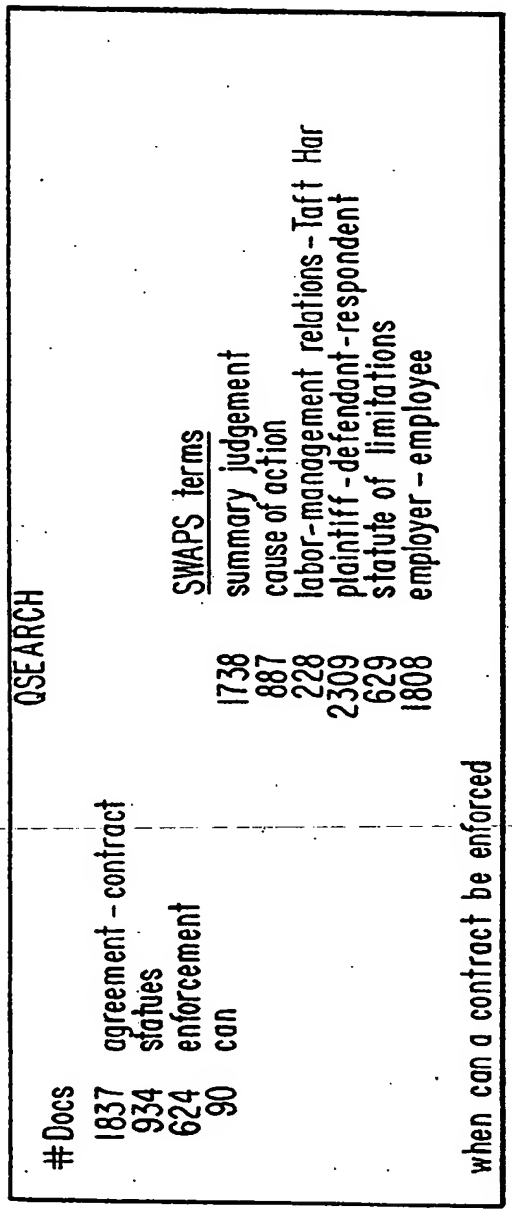

QSEARCH
#Docs
1837 agreement-contract
934 statues
624 enforcement
90 can
SWAPS terms
1738 summary judgement
887 cause of action
228 labor-management relations-Taft Har
2309 plaintiff-defendant-respondent
629 statute of limitations
1808 employer-employee
when can a contract be enforced

FIG.5

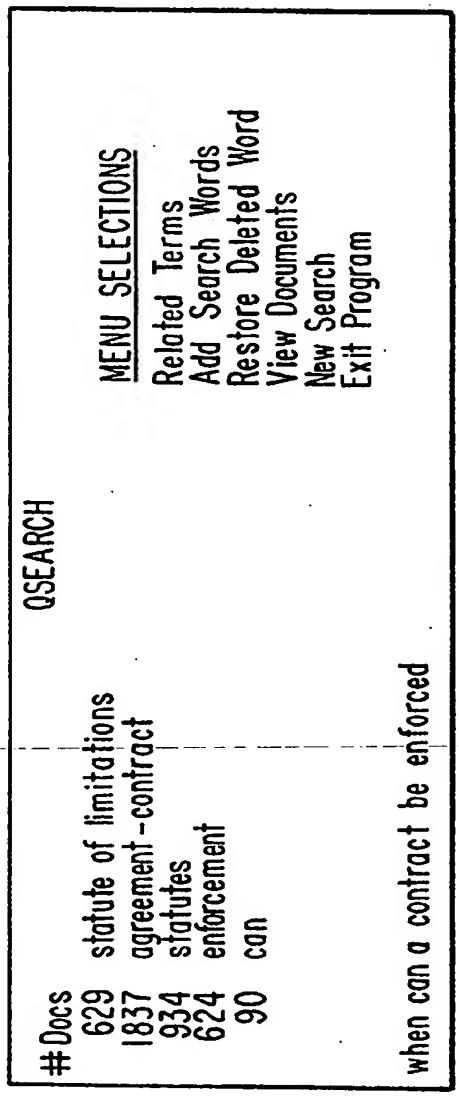

QSEARCH
#Docs
629 statute of limitations
1837 agreement-contract
934 statutes
624 enforcement
90 can
MENU SELECTIONS
Related Terms
Add Search Words
Restore Deleted Word
View Documents
New Search
Exit Program
when can a contract be enforced

FIG.6

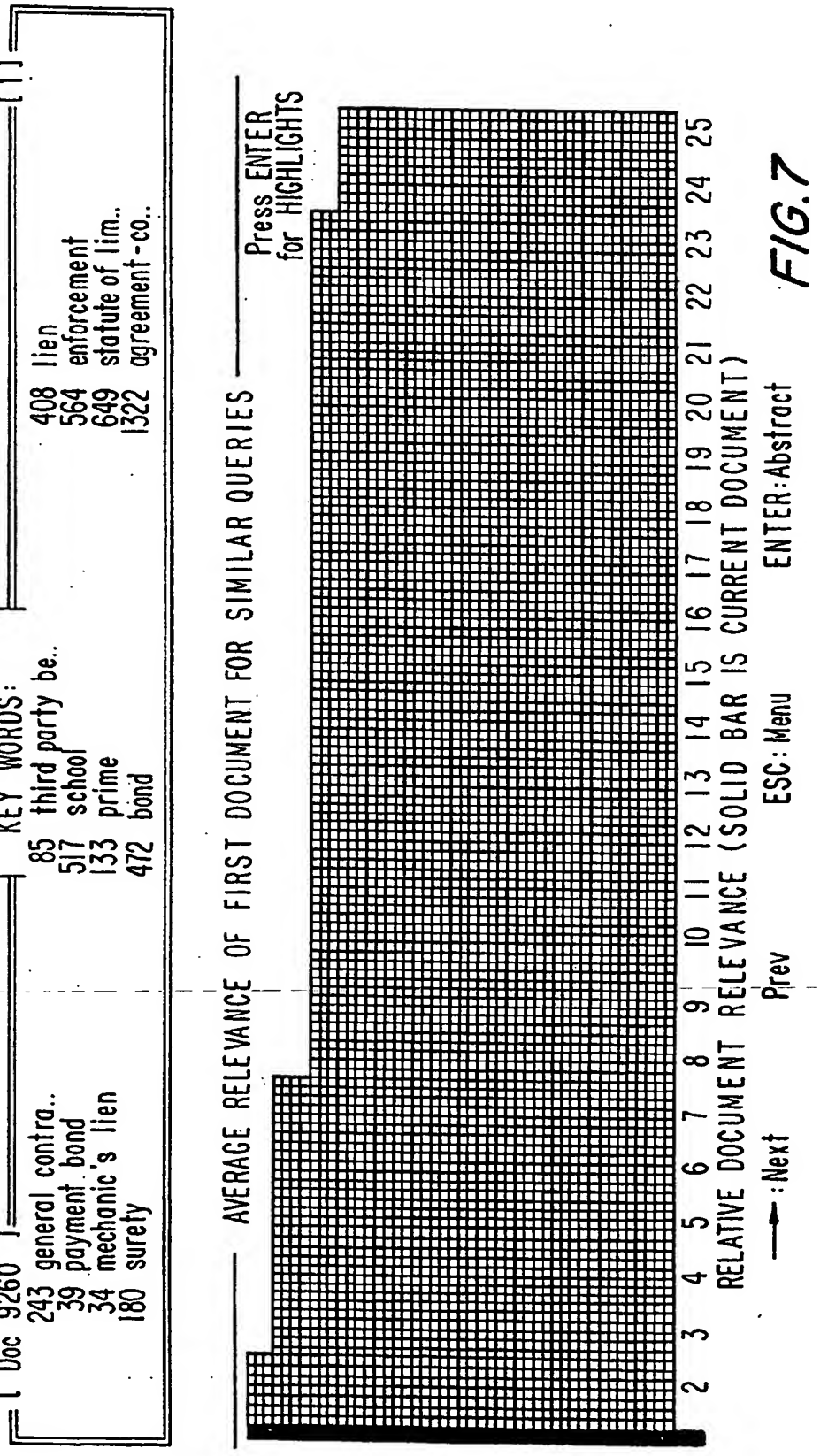

[Doc 9260]
243 general contra..
39 payment bond
34 mechanic's lien
180 surety
KEY WORDS:
85 third party be..
517 school
133 prime
472 bond
[ ]
408 lien
564 enforcement
649 statute of lim..
1322 agreement-co..
AVERAGE RELEVANCE OF FIRST DOCUMENT FOR SIMILAR QUERIES
Press ENTER for HIGHLIGHTS
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
RELATIVE DOCUMENT RELEVANCE (SOLID BAR IS CURRENT DOCUMENT)
:Next
Prev
ESC:Menu
ENTER:Abstract

FIG. 7

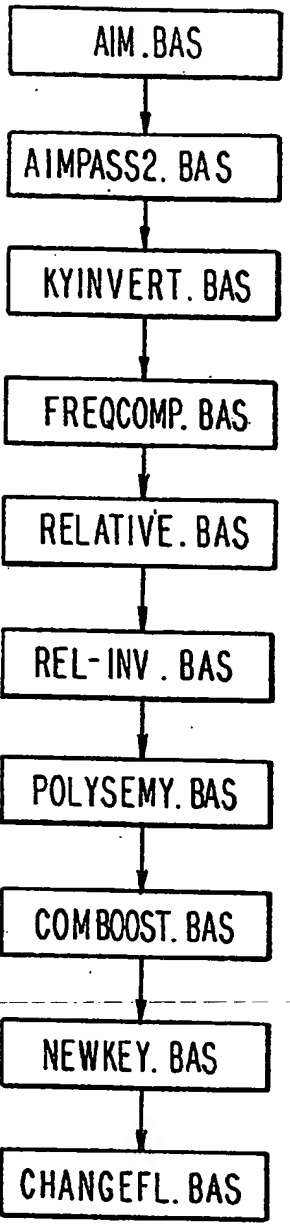


FIG.8

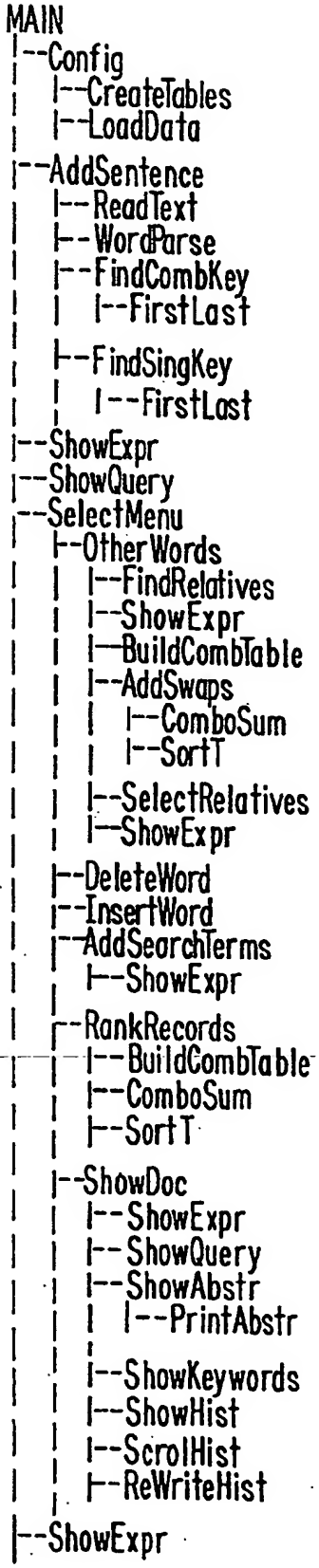


FIG.9

```
' AIM.BAS: Automatic Indexing Module
' Invoked: aim configfile machine# firstdoc lastdoc
' Creates: DocKeys. AH, DocIndex.AH, IDF.SIN,Stats#.Txt
' Uses: Doc.Txt, Doc.Ndx, SingKey.Str, CombKey1.Str, CombKey2.Str, CombKey3.Str
'       KeyWord.Tbl, KeyComb. Tbl
' DocKeys hold ALL of the Keywords and Counts for all documents
'   TYPE DocKeyType
'       Code AS INTEGER
'       Freq AS INTEGER
' DocNdx(x).NDX points at the element in DocKeys() where the Keys for
'   TYPE DocIndexType
'       Ndx AS LONG
'       Num AS INTEGER
'       Tot AS LONG
'       Pad AS STRING*6
' Document #x are, .NUM is how many keys are in the document,
' and. TOT is the total sum of all occurrences of keys
' Idf. Sin holds the inverse document frequency i.e. # docs key is in
'   INTEGER (Not a TYPE)
' Stats#.Txt (# = machine # supplied on command line)
```

FIG.10A

```
AIMPASS2.BAS
'Invoked: aimpass2 ConfigFile
'Creates: Key.Ndx, Weight.Ndx, \Aim\Command$.Dat
'Uses: Dict.Wrd, Idf.Sin, DocIndex.Ah, DocKeys.Ah
'nth Rec of Key.Ndx contains NumKeysinDocn followed by 63 codes
'  TYPE KeyNdx
'     Num AS INTEGER
'     Code(1 TO 63) AS INTEGER
'nth Rec of Weight.Ndx contains 63 Salton Weights computed with the following
'salton weight formula
'  TYPE WeightNdx
'    Weight (1 TO 63) AS SINGLE
'SALTON WEIGHT FORMULA
'                    Log2(FreqinDoc+1)*Log2((TotDocs*2)/DocsWithWord+2+2*TotDocs)
'Weight(Word)=-----------------------------------------------------
'                          Log2 (2 + TotalKWordsInDoc / 10)
```

FIG.10B

```
'KYINVERT.BAS
'Invoked: kyinvert ConfigFile
'Creates: kyinvert.Ndx,KyInvert.Dat
'Uses: Key.Ndx,Weight.Ndx & Dict.Wrd for NumKeys
'nth Rec of KyInvert.Ndx contains nth Code, ptr into KyInvert.Dat &
'NumDocsWith Word
'  TYPE NdxType
'     Code AS INTEGER
'     Index AS LONG
'     Num AS INTEGER
'pointed to record of KyInvert.Dat contains first DocWithWord & 1000*WeightInDoc
'  TYPE RecValue
'     Rec AS INTEGER
'     Value AS INTEGER
```

FIG.10C

```
'FREQCOMP.BAS
' Implements the Inverted Index access method along with the weighted values
' to calculate the frequent companions for each of the words used in the
' document collection.
' Invoked: freqcomp ConfigFile
' Creates: Freqcomp.127
' Uses: KyInvert.Ndx, KyInvert.Dat, Key.Ndx, Weight.Ndx
' nth Rec of FreqComp.127 contains nth code, Num of companions of this code &
' 127 pairs of (code, 100*weight)
'  TYPE Freqcomp127
'      Num AS INTEGER
'      code AS INTEGER
'      comp(1 TO 127) AS INTEGER
'      Value(1 TO 127) AS STRING*1
```

FIG. 10D

```
'RELATIVE.BAS
    '----are there any FreqComps for this keyword?
    '----we found the word in j's FCList
    '----look for the word itself in word j's FCList
    '----apply formula of (Lower*6 + Higher)/7
    '----sort in decreasing order, by value
    '----save first 63 as the relatives
'Invoked: relative ConfigFile
'Creates: Relative.63
'Uses: FreqComp.127 & Dict.Wrd for Numkeys
'nth Rec of Relative.63 contains nth Code, Num of relatives of this code &
' 127 pairs of (code, 100*weight)
'  TYPE FreqComp63
'      Num AS INTEGER
'      Code AS INTEGER
'      Comp(1 TO 63) AS INTEGER
'      Value(1 TO 63) AS STRING*1
```

FIG. 10E

```
'REL-INV.BAS
'Invoked: rel-inv ConfigFile
'Creates: Rel-Inv.Ndx, Rel_Inv.Dat
'Uses: Relative.63 & Dict.Wrd for NumKeys
'nth Rec of Rel-Inv. Ndx, contains nth code, ptr into Rel-Inv. Dat & NumRelsOfCode
'  TYPE NdxType
'       Code AS INTEGER
'       Index AS LONG
'       Num AS INTEGER
'pointed to record of Rel-Inv.Dat contains code of the first FreqComp list
'of nth code & the nth code's value in that list
'  TYPE RecValue
'       Rec AS INTEGER
'       Value AS STING *1
```

FIG.10F

```
'POLYSEMY.BAS
'Invoked: polysemy ConfigeFile
'Creates: PolySemy. Dat & PolyAvg. Dat, PolySemy. Lst
'Uses: Relative. 63, Rel_Inv.Ndx, Dict. Wrd
'nth Rec of PolySemy. Dat contains Poly Value of nth word calculated as follows:
'POLY FORMULA:
'Poly!=(Avg3!*/Avg3!/Avg20!)+Avg6!*(Avg6!/Avg□63!))^.5*(RelFreq/DocFreq)^.4
'  TYPE PolyType
'       Code AS INTEGER
'       Value AS SINGLE
'       Pos AS STRING *2
```

FIG.10G

```
'COMBOOST.BAS
'boosts poly of combined keywords by mutliplying by them 1.4
'Invoked: comboost ConfigFile
'Creates:
'Uses: PolySemy.Dat, DictSort.Cod
```

FIG.10H

```
'NEWKEY.BAS
'Invoked: newkey ConfigFile
'Creates: NewKey.Ndx, NewVal.Ndx
'Uses: Key.Ndx, Weight.Ndx, PoltSemy.Dat, KyInvrts.Ndx
'NewVal.Ndx contains new weights for Doc n sorted by new weight where
'  NewWeight=(Weight * Poly) ^ .125
'    TYPE WeightAvgNdx
'         Weight (1 TO 63) AS SINGLE
'         Mult AS SINGLE
'Mult is used to vary the number of sentences in the abstract program
'NewKey.Ndx contains the corresponding codes for Doc n i.e sorted by
'  their new weights
'  TYPE KeyNdx
'      Num AS INTEGER
'      Code (1 TO 63) AS INTEGER
```

FIG.10I

```
'CHANGEFL.BAS
'Invoked changefl ConfigFile
'Creates: KyInvrts.Ndx, Rel-Invs.Ndx
'  TYPE SmallNdxType
'       Index AS LONG
'       Num AS INTEGER
'Uses: KyInvert.Ndx, RelInv.Ndx
'  TYPE NdxType
'       Code AS INTEGER
'       Index AS LONG
'       Num AS INTEGER
'makes copies of KyInvert.Ndx & Rel-Inv.Ndx without the code field
```

FIG.10J

Main:	Put up Menu For User
Config:	Initializes Data & Variables
LoadData:	Open Files & Load Some Files into Memory
AddSentence:	Get User Query & Convert Terms Found to Codes
ReadText:	Replace Punctuation etc. in Query
WordParse:	Build Table of Query Words
FindCombKey:	Match Query Table Against Combined Term Vocabulary
FirstLast:	Use Index to Find Range to Check in Vocabulary
FindSingKey:	Match Query Table Words Against Single Word Vocabulary & Strip Prefixes to Find Stem
ShowExpr:	Display Matched Query Terms
ShowQuery:	Display Query
SelectMenu:	Get User Menu Choice
OtherWords:	Find & Display Related Terms & Get User Choices
FindRelatives:	Retrieves Relatives for Given Word
BuildCombTable:	Build CombTable of Relative Percentages Modified by Relative Document Freq's For All Pairs of Query Terms
AddSwaps:	Build Sorted (Ranked) Table of Swap Terms
ComboSum:	Calculate Multiplier For Swap & Document Values
SelectRelatives:	Get User Choices of Swap Terms
DeleteWord:	Delete Term From Query Table
InsertWord:	Add the Last Deleted Term Back Into Query
AddSearchTerms:	Display All Terms Beginning With A User Entered String of Letters & Get User Choices
RankRecords:	Build Sorted (Ranked) Tables of Documents
ShowDoc:	Get User Choice After Ranking Documents
ShowAbstr:	Get Abstract Text From File
PrintAbstr:	Display Abstract Text
ShowKeywords:	Display Highest Weight Terms in the Document Chosen
ShowHist:	Display Histogram Showing Relative Document Values
ScrolHist:	Scroll Histrogram Horizontally
ReWriteHist:	Scroll Histrogram Horizontally

FIG. 11

Abstract specs

Sentence ENDS ALGORITHM:

_ = space
U = upper case letter
l = lower case letter
X = any character
N = upper case noise word
# = number
E = . or ? or !

1)	E_[_]U	ENDS
2)	E_[_]l	NOTENDS
3)	E_N	ENDS
4)	_UE_U	NOTENDS
5)	AE	NOTENDS
	where A =	zB, ggf, inkl, vgl, grunds, ausschl, einschl, Kl, Bekl, Nr, Ger, BerGer, ff, subj, obj
6)	LEN<6	NOTENDS

Sentence RANKING ALGORITHM:

WORD WEIGHT Formula:
AbstractWordWeightValue=DocumentWeight_* (PolyValue^ .125)
SENTENCE VALUE Formula:
SentVal!= sum over keywords in sentence of AbstractWordWeightValue
FinalSentValue!= SentVal! /SQR (NumWordsinSent)

FIG. 12A

Sanitize Specs

GermanData:

```
DATA "\ion\ung\", "\heit\zip\satz\keit\", "\ionen\nzips\ungen\viele\"
DATA "\kosten\schaft\heiten\keiten\manche\einige\vielen\saetze\satzes\"
DATA "\pflicht\saetzen\manchem\einigen\mehrere\"
DATA "\schaften\mehreren\", "\pflichten\"
DATA "\er\ow\", "\mann\burg\berg\"
```

EnglishData:

```
DATA "\al\", "\ion\ity\als\", "\ment\ions\ency\ancy\ness\", "\ments\ional\ities\"
DATA "\mental\encies\ancies\"
DATA "\o\i\", "\er\rg\os\ey\tz\", "\man\son\ang\ham\bn\scn\", "\mann\tein\"
a$ = "\ein\eine\einem\einen\eines\diese\dieses\dieser\diesen\diesem\dieser"
b$ = "\kein\keine\keiner\keinen\keinem\keines\"
eng$ = "\a\an\any\this\such\no\several\many\"
```

FIG. 12B

Sanitizing ALGORITHM

IF UC word at beginning of sentence or after " . " or " : " or ")" THEN keep (Don't XXXXX) IF

1. noise word
2. combined keyword
3. Article, Articles, Art, Section, Sections, Sec., Paragraph, Subparagraph, Title, Title, Chapter, Chapters
4. matches a single keyword > 2 AND ends in : ment, ments, mental, ion, ions, ional, ity ities, ency, encies, ancy, ancies, ness, al, als
5. matches a single keyword > 3 AND preceded by: a, an, any, this, such, no, several, many
6. matches a single keyword > 9 AND ends in: er, man, mann, rg, i, o, son, os, ong, ey, ham ton, tein, scn, tz
7. matches a single keyword > 4 AND preceded by: the
8. matches a single keyword > 7

ELSE

XXXXX the word out

IF UC word NOT at beginning of sentence AND NOT after " . " or " : " or ")" THEN keep (Don't XXXXX) IF

2-8

ELSE

XXXXX the word out

IF U [U] - 1 [ ] THEN keep (Don't XXXXX)

FIG. 12C

METHOD OF INDEXING AND RETRIEVAL OF ELECTRONICALLY-STORED DOCUMENTS

This is a continuation of application Ser. No. 97/998,023, filed Dec. 29, 1992; which is a continuation-in-part of U.S. application Ser. No. 07/456,558, filed Dec. 26, 1989, both now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates generally to document storage and retrieval systems and more particularly to a method of indexing documents so that they can be retrieved in response to a query in order of their relevance to the query. It also permits, general query to be easily modified based on the content of the documents so that the new query will retrieve documents that are relevant to the original query.

2. Description of the Prior Art

Document retrieval based on indexing of the documents in a document data base is well known. Typically the documents are indexed by creating an index file which records the documents that each word is in. Then when the user inputs a query, the documents that contain one or more words of the query can be quickly identified. However, if the query consists of general words that are not terms of art, the query may produce unsatisfactory retrieval results by either producing few documents that are of interest to the user or producing many documents that are not interesting to the user or both.

SUMMARY OF THE INVENTION

A principal object of the present invention is to provide an improved method of indexing and retrieving documents which:

(A) allows a user to easily modify his query based on the content of the documents so that the new query will retrieve documents that are of interest to the user;

(B) accurately ranks the documents in order of relevance to the query; and

(C) allows the user to peruse the documents extremely quickly.

Another object of the present invention is to use the Soft Boolean Connector concept to adjust the number of hits (i.e., the number of query words that a document is credited with for ranking purposes) by giving less than a full hit to a word that often co-occurs with other query words. Another object of the present invention is to use the Soft Boolean Connector concept to adjust the number of hits (i.e. the number of query words that a word is credited with by virtue of its being related to those query words) for a possible suggested word by giving less than a full hit to a word that often co-occurs with the other query words.

These objects, as well as other objects which will become apparent from the discussion that follows, are achieved according to the present invention by the following steps (note: in the following the words "term" and "keyword" stand for both a single word and a phrase consisting of a group of words, e.g., "patent application".):

1. Indexing the documents by creating index files of which documents contain each term, how many times the term appears in the document, and how many documents each term appears in.

2. Assigning as many weights to each term as there are documents that contain that term, where the weight of a term in a document depends on the number of times the term appears in the document, the number of documents that the term appears in, and the total number of terms in the document.

3. Constructing for each term a ranked list of companions of said term which list contains the terms (companions) that appear in the same documents as said term in order of the sum of the weights of the companions over all documents that contain both the term and the companion. Associated with each companion is the companion percentage which is the sum used to rank the companions.

4. Using the companion lists to construct relative lists for each term which relative lists usually contain only those companions which also have said term as a companion. Associated with each relative is the relative percentage which is a weighted average of the companion's percentage as a companion of the term and the term's companion percentage as a companion of the companion. The relative percentages are used to rank the relatives.

5. Assigning a "polysemantic" weight to each term, which polysemantic weight depends on the number of documents that the term is in, the number of relatives that the term has, and the relative strength of the first few relatives to the other relatives.

6. Presenting to the user, in response to a query, a list of "SWAPS" (Synthetic Word Association Pattern Search) terms that are the best relatives to the entire group of terms contained in the query and allowing the user to add one or more of the presented terms to the query.

7. Ranking the documents according to how many query terms are contained in the document, their polysemantic weights and their weights in the documents.

The present invention facilitates the rapid searching of a document data base for documents that are of interest to the user. By using the suggested SWAPS terms the user can modify his query so as to retrieve those documents, if they exist in the data base, which are of interest. Since the SWAPS terms that are presented are in many of the documents that the original query terms are in, adding them to the query is guaranteed to retrieve those documents and others containing the SWAPS terms. By using the SWAPS feature repeatedly the user can in effect roam around the data base without actually retrieving and reading documents. Only after the query has been modified to include all the interesting SWAPS terms, does the user need to actually retrieve the documents. The user can start with a poor query and modify it using SWAPS so that it becomes a good query. The user need not waste time formulating a good query that will not retrieve any relevant documents because there happen to be no such documents in the data base. The SWAPS terms that are suggested will always retrieve documents that contain them i.e. documents that are likely to be relevant.

The ranking of the documents also facilitates rapid searching because the user can be confident that the highest ranked documents will be the documents that are most relevant to the query and that all documents which have any relevance will be retrieved and ranked.

The foregoing and other objects, features and advantages of the present invention will become apparent from the following, more particular description of the

preferred embodiments of the invention, as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a computer system embodying the present invention;

FIG. 2 is a view of the display screen showing an entered query and the result of parsing it;

FIG. 3 is a view of the display screen showing suggested SWAPS terms for the query of FIG. 2;

FIG. 4 is a view of the display screen showing the modified query;

FIG. 5 is a view of the display screen showing suggested SWAPS terms for the modified query of FIG. 4;

FIG. 6 is a view of the display screen showing a second modification of the query based on choosing SWAPS terms from FIG. 5;

FIG. 7 is a view of the display screen as a result of ranking the documents for the query of FIG. 6;

FIG. 8 is an operational flow diagram for indexing a set of documents;

FIG. 9 is a procedure tree for the QSEARCH program used for searching an indexed set of documents using the SWAPS and RANKING features;

FIGS. 10A to 10J are description of the program modules in FIG. 8;

FIG. 11 is a description of the program modules in FIG. 9; and

FIGS. 12A to 12C are description of the ABSTRACT program module.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

This invention will now be described as embodied in a computer system of the type shown in FIG. 1. This embodiment utilizes the following computer hardware and software:

(1) IBM compatible personal computer with at least 4 MB of RAM, a large capacity hard drive, a display screen, and a keyboard.

(2) MS-DOS compatible operating system and LIM 3.2 compatible expanded memory manager.

(3) A vocabulary file of terms (words and phrases)

(4) A series of programs that index the documents by constructing various files that hold information about which terms are in which documents, which documents contain which terms, the weights of the terms, and which terms are relatives of other terms by virtue of occurring in the same documents and how strongly are they are related.

(5) A user program that accepts a query, suggests modifications to the query, and ranks the documents based on the modified query using the weights and relative strengths of the terms of the query.

The Vocabulary file is structured as a list of headwords each with a short synonym list. All of the synonyms of a given headword are assigned the same code.

The full list of indexing programs can be found in FIG. 10. Here we will describe the most important of these programs: AIM, AIMPASS2, FREQCOMP, RELATIVE, and POLYSEMY.

The first indexing program is AIM.BAS: Automatic Indexing Module. It creates DocKeys, DocIndex, and IDF. DocKeys holds all of the Keywords and Keyword-Counts for all documents. IDF holds the document frequency, i.e., the number of documents a keyword appears in.

As the words in the documents are checked against the vocabulary to see if they are keywords, the case (upper or lower) is possibly changed and they are stripped of prefixes to see if the different case or stem is a keyword according to the following algorithms: (UC=upper-case and LC=lower-case)

IF UC word is at the beginning of a sentence AND we don't have it in our vocabulary as a LC word THEN look for it the Vocabulary as an UC word

IF UC word in middle of sentence AND we don't have it UC THEN look for it if it doesn't have a typical proper name ending

In USER Program Only: IF word NOT found THEN find both the stem AND find the Good prefix (In the following "find" means that the stem and/or prefix is said to be in the document if the prefix is of the right type and the stem has the indicated length and is a keyword.)

IF GOOD prefix THEN
Find GOOD prefix if stem>3 characters long
IF word is found THEN find if stem >8 characters long
IF word is NOT found THEN find if stem >5 characters long
IF POOR prefix THEN
If word is found THEN DON'T find stem
If word is NOT found THEN find if stem >5 characters long

List of Poor Prefixes:

hi, co, de, en, ex, im, in, un, re, con, eco, dis, epi, mal, mid, mis, non, off, out, pre, pan, sub, uni, demi, down, fore, hemi, high, meta, over para, peri, post, self, semi, after, inter, quasi, trans, under

List of Good Prefixes:

air, bio, sea, sky, top, aero, anti, auto, back, head, home, homo, hemo, mega, mini, mono, rear, poly, self, tele, viro, chemo, ferro, homeo, hyper, infra, intra, macro, micro, multi, hydro, radio, super, supra, ultra, contra, hetero, thermo, techno, nucleo, counter, electro, magneto

The next indexing program is AIMPASS2.BAS. It creates Key and Weight files. The nth Rec of Key.Ndx contains NumKeysinDoc(n) followed by up to 127 Key codes which have Weight greater than or equal to the Adaptive Threshold Value. The Adaptive Threshold Value is the average Weight value of the 80th Keyword in each document (0 if there are less than 80 Keywords in a document). The nth Rec of Weight. Ndx contains up to 127 (or as many Keywords are above the Adaptive Threshold Value) Document Weights computed with the following weight formula:

$$\text{Weight(Word)} = \frac{\text{Log}_2(\text{FreqInDoc} + 1) \times \text{Log}_2\left(\frac{\text{TotDocs} \times 1.5}{\text{DocsWithWord} + 3 + \frac{\text{TotDocs}}{1000}}\right)}{\text{Log}_2\left(2 + \frac{\text{TotalKeywordsInDoc}}{10}\right)}$$

FREQCOMP.BAS implements the Inverted Index access method along with the weighted values to calculate the frequent companions for each of the words used in the document collection.

For each word ("A") in the controlled vocabulary dictionary, the WEIGHT (see above formula) values for each co-occurring word in the document (a co-occurring word to A is one that appears as a Keyword in the same document that A appears as a Keyword) are summed, along with the WEIGHT values for A in that document, respectively in all documents in which they co-occur. The sum values for each co-occurring word are converted to a percentage, scaled to the sum value for A (i.e., percentage=sum for word's WEIGHT values divided by the sum for A's WEIGHT values). Note that the percentages for the co-occurring words can be higher than 100% if they are heavily weighted in the same records in which A appears. The co-occurring words are then sorted in descending order (from highest percentage value to lowest) and the top 127 are written to a file (see below for structure). If there are 127 co-occurring words or fewer, then all of the co-occurring words will be written in descending sorted order..

Definitions:
{} = co-occurring
weight = WEIGHT value

Example:

Doc# 1	Doc# 2	Doc# 3	Doc# 4	Doc# 5
A 0.5	A 1.5	A 2.0	A 0.7	A 0.8
B 0.8	.	B 1.9	B 1.2	B 2.5
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

A {} B C D E ...
B {} A C D E ...

5.5 = sum of weight of all A's
6.4 = sum of weights of B's co-occurring with A

Sample:

A 5.5 {} B 6.4/116% ...
B 6.4 {} A 4.0/63% ...

Resulting File:

Main Word	Co-Occurring Words . . . (sorted)
A	B 116% . . .
B	A 63% . . .
.	
.	
.	

After the frequent companions have been found RELATIVE.BAS is run to define the relatives of each Keyword (A) according to the following algorithm:

are there any FreqComps for A? If so, then for each FreqComp of A (F):
look for F in A's FreqComp List and get its value
look for the word itself (A) in word A's FreqComp List and get its value
apply formula of (Lower×6+Higher)/7, where Lower is the lower of the two values obtained in the above two steps and Higher is the higher of the two values.
a sort in the resulting list of words and values in decreasing order, by value
a save the first 63 (or as many as are found) of this list as the relatives for keyword A

For each word (called "A") in the dictionary which has Frequent Companions (not all do, because some words in the dictionary are not used at all in a database), take each Frequent Companion of A (called "F") and its Frequent Companion Percentage Value [FCPVal] in A's Frequent Companion List [FCList](called "F-VAL") and look for the FCPVal of A in F's FCList (called "A-VAL"). NOTE: If A is not found in F's FCList, then A-VAL is zero (0). The RELATIVE value for F is calculated by multiplying the smaller of F-VAL and A-VAL by 6, adding the larger of F-VAL and A-VAL, and then dividing that sum by 7. If both A and F are in each other's FC lists, the resulting Relative value will be added to both words' Relative lists. If F is in A's FC List, but A is not in F's then F-VAL will be divided by seven and added only to A's Relative list.

After all the RELATIVE values are calculated for each Frequent Companion (F) in A's FCList, they are sorted in descending order and the top 63 of these words are written to A's Relative List. If there are fewer than 63 Relatives, then all of the Relatives will be written to A's Relative List, in descending order of RELATIVE value.

(based on above Frequency Companion File)
Formula:

$$\text{Relative Value} = \frac{(\text{SmallerPercent Value} \times 6) + \text{LargerPercent Value}}{7\ (12 \text{ if not mutual})}$$

Here the SmallerPercent Value is the smaller of the A-VAL and the F-VAL and the LargerPercent Value is the larger of the A-VAL and the F-VAL.

Sample:

$$70 = \frac{(A_{(63\%)} \times 6) + B_{(116\%)}}{7}$$

Resulting File:

Main Word	Relatives . . . (sorted)
A	B 70 . . .
B	A 70 . . .
.	
.	
.	

After the relatives have been found each of the keywords is given a single polysemantic weight that does not change from document to document by the program POLYSEMY.BAS which uses the following formula:

$$\text{PolyValue} = \sqrt{\left(\text{Avg}_3 \times \frac{\text{Avg}_3}{\text{Avg}_{20}} + \text{Avg}_6 \times \frac{\text{Avg}_6}{\text{Avg}_{63}}\right) \times \sqrt{\frac{\text{TotRelVal}}{\text{DocFreq}^{1.2}}}}$$

Here Avg_n is the average of the relative percentages of the first n relatives of the keyword, TotRelVal is the sum of relative percentages over all relative lists that the keyword is in, and DocFreq is the number of documents that the keyword is in (having a WEIGHT above the adaptive threshold).

Once the indexing programs have been run, the ABSTRACT program is run to create highlights of the full

text that will be presented to the user before or in place of the full text itself. First the documents are broken into sentences using a Sentence Ends Algorithm. Then the sentences are assigned weights (values) as a whole and the top ranked sentences are chosen to be part of the highlight. Finally a Sanitize algorithm is used to "X" out (eliminate) proper names from in the highlights. See FIG. 12 for specific details on the algorithms used in the ABSTRACT program.

Once the indexing and optionally the ABSTRACT programs have been run, the QSEARCH program can be used to search for documents. This is done by entering a query in natural language. The user program will parse the query to find all the keywords it contains using algorithms similar to those in the AIM program.

After the query is parsed the user is shown the keywords that are contained in the query in order of their polysemantic weight and is given the opportunity to add and delete words in the query and to have the program suggest SWAPS terms based on the query. These SWAPS terms are generated by generating for each keyword in the vocabulary a summed-relpoly-percentage which is the sum, over all terms that are in the query, of the relpoly percentages of that keyword, where the relpoly percentage is the product of the relative percentage and the polysemantic weight. Then the summed RelPoly percentages are adjusted using a concept called Soft Boolean Connectors to come up with a final SWAPS value for each keyword. The keywords are then ranked by SWAPS value and the highest ranked are presented to the user as suggested SWAPS terms to be added to the query.

The Soft Boolean Connectors concept involves penalizing pairs of terms that co-occur often (i.e., in many documents) when calculating the adjustment to be applied to the summed relpoly percentages.

First, Multiply the last group of SWAPS words by
Boost Factor (=2)
Then add relative values of relatives of
main word after each is multiplied by the
PolyValue of the Word
(The previous value will be called "Temp Value")
Create table for every pair combination of query words, e.g.,
for words A, B, & C, there are three pairs:
AB
AC
BC

For each pair of query words, ("A" & "B"), the Relative Value used in the formula below is B's Relative Value in A's Relative List, or, if B doesn't appear in A's Relative List, then the value is taken from A's Relative Value in B's Relative List (this is possible because the Relative Value between any two words is mutual), i.e., if B is found in A's Relative list, take just that value. You don't need to look at B's list to find A's value there because, if it is there, it would have the same value as B has in A's list. Only if B is not in A's Relative list check for A in B's list. Enter the Relative Penalty value resulting from the following formula into the table for each combination (pair):

$$\frac{\text{Relative Value} \times \sqrt{\text{avg doc freq of the respective combination (e.g., A \& B)}}}{\sqrt{\text{avg doc freq of all words in the database}}} \Big/ 42$$

-continued

Example:
if Relative values are
AB = 70
AC = 75
BC = 65
and A appears in 5 documents and B appears in 4 and the avg. doc freq is 4, then using the following formula for AB

$$1.77 = \frac{70 \times \frac{\sqrt{4.5}}{\sqrt{4}}}{42}$$

the table value for AB is 1.77
(look below for maximum, making this 1.0 instead)

for a hit of 3 words - 3 relatives of main word or
3 words in a document
A B C
Adjust Value = $2.5^{(\#\ of\ hits - sum\ of\ penalties)}$

Note: The "# of hits" value is:

$$\sum_{i}^{n} \frac{\sqrt[4]{\text{PolyValue of word}}}{\sqrt[4]{\text{Avg PolyValue}}}$$ where n is 3 in this example

Word/Document Value = Temp Value × Adjust Value

MAXIMUM PENALTY TABLE (SWAPS)

query words	Max.
(for each pair)	
2	0.3
3	1.0
4 & up	0.9
(for sum of pairs)	
2	0.3
3	1.4
4	1.8
5	2.3
6 & up	2.8

After the user has modified the query by choosing SWAPS terms, he can have the program suggest new SWAPS terms based on the new query. In this case the program boosts the relative percentages of the last chosen set of SWAPS terms before calculating summed relpoly percentages. This allows the user to navigate in the data base by modifying his query so that it will find documents containing the SWAPS terms.

For example, FIG. 2 shows the options the user will be presented with after entering the query "when can a contract be enforced". If the user chooses the menu option "Related Terms" he will be shown a list of SWAPS terms as shown in FIG. 3. This first set of SWAPS terms that are presented to the user includes the term "statutes". The user may choose one or more of these suggested SWAPS terms to add to the query. In FIG. 4 we see that the user has chosen to add the term "statutes" to the query. At this point the user can again ask the system to suggest SWAPS words. This time the previously added SWAPS term "statutes" will be given extra weight in determining which new terms are suggested to the user. In FIG. 5 we see the resulting suggested SWAPS terms generated from the four query terms "agreement", "statutes", "enforcement", and "can", with "statutes" given more weight than the other three terms. Notice that the SWAPS words are ranked somewhat differently than in FIG. 3 and in

particular a new SWAPS term "statute of limitations" is suggested. By adding the term "statutes" to the query and then asking again for suggested SWAPS terms the user has "moved" the query to "an area of the database" that contains documents dealing with "statute of limitations", which is a term of art that makes the original query more focused and is likely to find documents that are relevant to the intent of the original query. Here the fact that both terms "statutes" and "statute of limitations" contain the same word is fortuitous. It is the meaning of the term "statutes" which makes it a close relative of "statute of limitations" by virtue of the fact that these two terms co-occur in many of the same documents.

Once the user is satisfied with his query he asks the program to retrieve documents that are relevant to the query. In FIG. 6 he would choose the View Documents option. The system will then use its index files to assign a value to each document and then rank the documents. The documents are ranked by generating for each document a summed-weightpoly-value which is the sum, over all terms that are in the query, of the weightpoly values of that keyword, where the weightpoly value is the product of the weight of the keyword in that document and its polysemantic weight. Then the summed-weightpoly values are adjusted using the Soft Boolean Connectors concept to come up with a final value for each document. The documents are then ranked by value and presented to the user in order of rank.

The Soft Boolean Connectors concept involves penalizing pairs of terms that co-occur often (i.e. in many documents) when calculating the adjustment to be applied to the summed relpoly percentages. First, multiply original query words by

Boost Factor (=2)

Then add WEIGHT values of key words in a document after each is multiplied by the PolyValue of the word.

(The previous value will be called "Temp Value")
Create table for every pair combination of query words (A B C)
AB
AC
BC

For each pair of query words, ("A" & "B"), the Relative Value used in the formula below is B's Relative Value in A's Relative List, or, if B doesn't appear in A's Relative List, then the value is taken from A's Relative Value in B's Relative List (this is possible because the Relative Value between any two words is mutual), i.e., if B is found in A's Relative list, take just that value. You don't need to look at B's list to find A's value there because, if it is there, it would have the same value as B has in A's list. Only if B is not in A's Relative list check for A in B's list. Enter the Relative Penalty value resulting from the following formula into the table for each combination.

Example:
if Relative values are
AB = 70
AC = 75
BC = 65
and A appears in 5 documents and B appears in 4
and the avg doc freq is 4
then using the following formula for AB

-continued

$$2.65 = \frac{70 \times \frac{\sqrt{4.5}}{\sqrt{4}}}{28}$$

the table value for AB is 2.65
(look below for maximum, making this 1.0 instead)
for a hit of 3 words - 3 words in a document
A B C
Adjust Value = 1.8(*# of hits − sum of penalties*)
Note: The "# of hits" is the same as above for the SWAPS.
Word/Document Value = Temp Value × Adjust Value

MAXIMUM PENALTY TABLE (RANKING)

query words	Max.
(for each pair)	
2	0.5
3	1.3
4	1.2
5 & up	1.1
(for sum of pairs)	
2	0.5
3	1.6
4	1.9
5	2.3
6 & up	2.8

To facilitate very rapid perusal of the ranked documents, the document values (used in the ranking) are presented as a bar graph as shown in FIG. 7. Also the documents are presented in 3 forms. The first form consists of a ranked array of the highest ranked terms in the document that requires only about ⅛ of the display screen (FIG. 7). The second form consists of a program generated "highlight" of the document which consists of very short portions of the document of less than a dozen words that contain the highest ranked terms. This highlight scrolls in about ⅜ of the screen and is shown along with the array of highest ranked terms. The third form consists of the full text of the document which can be scrolled. The user can use arrow keys to move rapidly from one document to the next.

Appendix 1 contains the full BASIC program source code that implements the preferred embodiment described above. This code must be compiled using the Microsoft 7.1 BASIC compiler to produce object modules which must then be linked along with libraries containing object code for assembler routines from the Crescent Software QuickPak Professional Advanced Programming Library for BASIC Compilers Version 4.12 to produce an executable file.

There has thus been shown and described a novel document indexing and retrieval system which fulfills all the objects and advantages sought therefor. Many changes, modifications, variations and other uses and applications of the subject invention will, however, become apparent to those skilled in the art after considering this specification and the accompanying drawings which disclose the preferred embodiments therefor. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by the invention which is limited only by the claims which follow.

APPENDIX 1: PROGRAM LISTINGS

```
DEFINT A-Z
'-------------------------------------------------------------------------
'          AIM: Automatic Indexing Module $Revision:  8.05 VN $
'-------------------------------------------------------------------------
'check U-case word against Lease also (CheckUpperCaseWord)

'$INCLUDE: '\USER\INCLUDE\TYPES.BI'

TYPE CodeType
        Code AS INTEGER
END TYPE

TYPE DocIndexType
        Ndx AS LONG
        Num AS INTEGER
        'Tot AS INTEGER     '-- Total Number of keys in doc for DJ (shorter docs)
        Tot AS LONG         '-- Total Number of keys in doc for WEST (Longer docs)
        Pad AS STRING * 6 '-- need to pad it out to 16 bytes for > 12,800 docs
                                                '   when using Tot as LONG
END TYPE

TYPE DocKeyType
        Code AS INTEGER
        Freq AS INTEGER
END TYPE

TYPE Str47
        Str AS STRING * 47
END TYPE

TYPE Str18
        Str AS STRING * 18
END TYPE

TYPE NewCombType
        Str AS STRING * 45
        Code AS INTEGER
END TYPE

TYPE NewSingType
        Str AS STRING * 16
        Code AS INTEGER
END TYPE
DECLARE SUB CheckUCaseWord (Word$, PrevWord$, Length%, NoNameFlag%)
DECLARE SUB Config (Machine$, First&, Last&)
DECLARE SUB DispMsg (Msg$, r%, c%)
DECLARE SUB DispStat (s$)
DECLARE SUB EmsAlloc (NumPages%, Handle%, LoadFILE$)
DECLARE SUB FindCombKey (WordEMS%, NumWords&, KyEMS%, NumKy%, CombKeyEMS%, NumCombKey%)
DECLARE SUB FindSingKey (WordEMS%, NumWords&, KeyEMS%, NumKey%, SingFoundEMS%, NumSingFound&)
DECLARE SUB pause (Tika%)
DECLARE SUB ReadEnglishText (FirstLine&, LastLine&, Handle%, Lin%)
DECLARE SUB ReadGermanText (FirstLine&, LastLine&, Handle%, Lin%)
DECLARE SUB ReadSection (Txt$, SecArray$(), ArtArray$())
DECLARE SUB WindMgr (ULRow%, ULCol%, LRRow%, LRCol%, Frame%, BoxColr%, TextColr%, Text$)
DECLARE SUB WordParse (TextHandle%, Lines%, WordHandle%, Words&)
DECLARE FUNCTION FirstLast% (Word$, First%, Last%, KeyType%)
DECLARE FUNCTION LoadIntoEMS% (File$)
DECLARE FUNCTION Num$ (x%)
DECLARE FUNCTION XLate% (x$)

'-------------------------------------------------------- External routines

'$INCLUDE: '\USER\INCLUDE\DECLARES.BI'

'-------------------------------------------------------- PROGRAM START

CONST Sing = 0, Comb = 1

COMMON SHARED NormAttr, RevAttr, File&, SecCode, ArtCode
COMMON SHARED DocDir$, LstDir$, AtList$, Lang$
COMMON SHARED XLateTable%(), SingTable%(), CombTable%(), NLGS(), LENNLG()
COMMON SHARED ThirtyTwo, SixtyFour, SixteenK, ThirtyTwoK&
COMMON SHARED ASCEND, DESCEND, FALSE, TRUE
COMMON SHARED ENTER, ESCAPE, ASCa, ASCz, ASCupperA, ASCupperZ, ASCslash, ASC0, ASC9, ASCat
COMMON SHARED CodeTemp AS CodeType, LENCode, LenComb, LenSing
COMMON SHARED NumComb1Keyword, NumComb2Keyword, NumComb3Keyword
COMMON SHARED Comb1KeywordEMS%, Comb2KeywordEMS%, Comb3KeywordEMS%
COMMON SHARED NumSing1Keyword, NumSing2Keyword
COMMON SHARED Sing1KeywordEMS%, Sing2KeywordEMS%, Prefixes$(), MeanPrefixes$()
COMMON SHARED Numbers$(), Section$(), Article$(), Paragraph$(), Artikel$()
COMMON SHARED NoNameEnding$(), NameEnding$()
NLGData:
DATA "mr","mrs","messrs","sen","rep","ms","dr","drs"

EnglishData:
DATA "Section","Sec","Sec."
DATA "Article","Art.","Art"

GermanData:
DATA "Paragraph","Par","Par.","Para."
DATA "Artikel","Art","Art."

NumData:
DATA "I","II","III","IV","V","VI","VII","VIII","IX","X"
DATA "XI","XII","XIII","XIV","XV","XVI","XVII","XVIII","XIX","XX"
DATA "XXI","XXII","XXIII","XXIV","XXV","XXVI","XXVII","XXVIII","XXIX","XXX"
```

```
NameEnd:
DATA "\i\o\","\er\rg\os\ey\tz\","\aan\son\ong\han\ton\scn\","\ann\tein\"
NoNameEnd:
DATA "\al\","\ion\als\ity\","\ment\ions\ency\ancy\ness\"
DATA "\ents\ional\ities\encies\ancies\","\ental\"
'$INCLUDE: '\user\include\prefpat.bi'

'------------------------------------------------------------------
Config Machine$, FirstDoc&, LastDoc&
Rev$ = "$Revision:   5.12 $"
Rev$ = MID$(Rev$, 14, 7)
a12! = 0
a14! = 0
CLS
WindMgr 2, 2, 24, 79, 2, NormAttr, RevAttr, "Automatic Indexing Module (Rev " + Rev$ + ")"

'------------------------------------------------ Read in Combined Keys

IF Lang$ = "GERMAN" THEN
        LenComb = 64 '62+2
ELSE
        LenComb = 47  '45+2
END IF


'it had to divide combkey.str to 3 files because array can't be > 128k
'and there is no memory to make element size power of 2

LoadFILE$ = LstDir$ + "COMBKEY1.STR"
IF NOT ExistX(LoadFILE$) THEN CLS : PRINT LoadFILE$; " not found.": END
NumComb1Keyword = FileSize&(LoadFILE$) \ LenComb
DispMsg "Loading " + Num$(NumComb1Keyword) + " Combined Keywords", r, c
Comb1KeywordEMS% = LoadIntoEMS%(LoadFILE$)
DispMsg "", r, c

LoadFILE$ = LstDir$ + "COMBKEY2.STR"
IF NOT ExistX(LoadFILE$) THEN CLS : PRINT LoadFILE$; " not found.": END
NumComb2Keyword = FileSize&(LoadFILE$) \ LenComb
DispMsg "Loading " + Num$(NumComb2Keyword) + " Combined Keywords", r, c
Comb2KeywordEMS% = LoadIntoEMS%(LoadFILE$)
DispMsg "", r, c

LoadFILE$ = LstDir$ + "COMBKEY3.STR"
IF NOT ExistX(LoadFILE$) THEN CLS : PRINT LoadFILE$; " not found.": END
NumComb3Keyword = FileSize&(LoadFILE$) \ LenComb
DispMsg "Loading " + Num$(NumComb3Keyword) + " Combined Keywords", r, c
Comb3KeywordEMS% = LoadIntoEMS%(LoadFILE$)
DispMsg "", r, c

'------------------------------------------------ Read Single Keys
LoadFILE$ = LstDir$ + "SINGKEY.STR"
IF NOT ExistX(LoadFILE$) THEN
        CLS
        PRINT "AIM Fatal Error:"
        PRINT
        PRINT LoadFILE$; " not found."
        PRINT
        chime 10
        PRINT "Press the SPACE BAR to exit:"
        i$ = INPUT$(1)
        END
END IF

DIM SingKeyTemp AS STR32
NumKeyword = FileSize&(LoadFILE$) \ ThirtyTwo

DispMsg "Loading " + Num$(NumKeyword) + " Single Keywords", r, c

SingKeywordEMS% = LoadIntoEMS%(LoadFILE$)

DispMsg "", r, c


'------------------------------------------------ Read 3-Char Tables

Symb = 28: First = 1: Last = 2
REDIM XLateTableX(38 TO 122)
REDIM SingTableX(1 TO Symb, 1 TO Symb, 1 TO Symb, 1 TO 2)
REDIM CombTableX(1 TO Symb, 1 TO Symb, 1 TO Symb, 1 TO 2)

XLateTable(47) = 1' / char, as used in non-wildcard words
XLateTable(38) = 2' & char, as used in S&P, A&P, etc.
FOR i = ASCa TO ASCz
        XLateTable(i) = i - 94' so that a=3, b=4,...,z=28
NEXT

FGetAH LstDir$ + "KEYWORD.TBL", SEG SingTableX(1, 1, 1, 1), (4 * 28), (28 * 28)
FGetAH LstDir$ + "KEYCOMB.TBL", SEG CombTableX(1, 1, 1, 1), (4 * 28), (28 * 28)

'------------------------------------------------ Open Documents

'-- actual count for total number of documents
Count& = FileSize&(DocDir$ + ".NDX") \ 8

OPEN DocDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS #8 LEN = 8
OPEN DocDir$ + ".TXT" FOR RANDOM ACCESS READ SHARED AS #9 LEN = 80

'DocKeys hold ALL of the Keywords and Counts for all documents
'DocNdx(x).NDX points at the element in DocKeys() where the Keys for
'Document #x are, .NUM is how many keys are in the document,
'and .TOT is the total sum of all occurrences of keys
```

```
'store ems for dockey and docndx-no need anymore because we write to disk each doc.
'4096 elements per 16k EMS page (4 bytes per element)
'NumPages = 600' MaxDocKeys& \ 4096&
'MaxDocKeys& = CLNG(NumPages) * SixteenK \ 4
'QPrintRC STR$(NumPages) + " EMS pgs for DocKeys.", 4, 2, -1
'QPrintRC "MaxDocKeys:" + STR$(MaxDocKeys&), 4, 59, -1
'EmsAllocMem NumPages, DocKeysEMS%
'IF EmsError% THEN PRINT "Couldn't allocate"; CLNG(NumPages) * SixteenK; "bytes of EMS for DocKeysEMS.": Chime 2: STOP
'EMSNdxPages = 20 '360k can hold 36000 doc
'EmsAllocMem EMSNdxPages, DocNdxEMS%
'IF EmsError% THEN PRINT "Couldn't allocate"; CLNG(EMSNdxPages) * SixteenK; "bytes of EMS for DocNdxEMS.": Chime 2: STOP

CurrDocKey& = 1       ' current pointer into DocKeys()
Debug = FALSE
Start& = FirstDoc&
'-- only use the LastDoc& parameter if there are at least
'   that many documents in the database, otherwise use the
'   actual number of documents
IF Count& > LastDoc& THEN Count& = LastDoc&

'-- create the DOCKEYS.AH file for output
FCreate LstDir$ + "DOCKEYS.AH" + Machine$
FOpen LstDir$ + "DOCKEYS.AH" + Machine$, DocKeyFile%
IF DocKeyFile% = -1 THEN
         PRINT "Can't create DOCKEYS.AH"
         chime 2
         END
END IF


FCreate LstDir$ + "DOCINDEX.AH" + Machine$
FOpen LstDir$ + "DOCINDEX.AH" + Machine$, DocNdxFile%
IF DocNdxFile% = -1 THEN
         PRINT "Can't create DOCINDEX.AH"
         chime 2
         END
END IF

'AHSavePoint& = 1  '-- intermediate save point
'NdxPoint = Start&

DIM DocNdx AS DocIndexType
LenDocNdx = LEN(DocNdx)

DIM DictTemp AS DictType  'only for getting size dependening only on DictType
DictWordNum = FileSize&(LstDir$ + "DICT.WRD") \ LEN(DictTemp)

REDIM Idf&(1 TO DictWordNum) ' total number of word-codes  idf shows numofdoc where this word is
IF NOT Debug THEN
       '-- open a new file for statistics
       OPEN LstDir$ + "STATS" + Machine$ + ".TXT" FOR APPEND SHARED AS #5
       PRINT #5, "-------------------------------------------------"
       PRINT #5, "Start Time:"; TIME$; TStart!
       PRINT #5, ""
ELSE
       OPEN "\DEV\NUL" FOR OUTPUT AS #5
END IF

Proc = 0' number of files actually processed
TStart! = TIMER
MaxTempKey = 1000  'number of unique keys in one doc. if more then redim preserve

DIM Doc AS ISAMtype
CodeString$ = ""
FreeStrSp& = FRE("")

'************************* S T A R T ***************************************

FOR File& = Start& TO Count&

REDIM DocKeyTemp(1 TO MaxTempKey) AS DocKeyType
LENDocKeyTemp = LEN(DocKeyTemp(1))

PRINT #5, "File #"; MID$(STR$(File&), 2)

GET #8, File&, Doc
NumLines = Doc.Last - Doc.First + 1'  lines of text in the file

'---- Allocate EMS to hold the Text file
NumPages = 80& * NumLines / SixteenK + 1' 80 bytes per line, 16k per EMS page
CALL EmsAllocMem(NumPages, TextEMS%)
IF EmsError% THEN PRINT "Couldn't allocate"; NumPages * SixteenK; "bytes of EMS.": STOP

'---- Allocate EMS for the parsed word array (32 bytes per word)
NumPages = CLNG(NumLines) * 12 \ 512 + 1' max number of words (512 per 16K EMS page)
CALL EmsAllocMem(NumPages, WordEMS%)
IF EmsError% THEN PRINT "Couldn't allocate"; NumPages * SixteenK; "bytes of EMS.": STOP

Row = 6
CALL WindMgr(Row, 30, Row + 1, 50, 2, RevAttr, RevAttr, "CURRENT DOCUMENT")
QPrintRC MID$(STR$(File&), 2) + "/" + MID$(STR$(Count&), 2), Row + 1, 35, -1

'---------------------- Read Document into EMS eliminating blank lines
'                        strip non-alpha chars and crunch extra spaces

IF Lang$ = "ENGLISH" THEN
       ReadEnglishText Doc.First, Doc.Last, TextEMS%, NumLines%
ELSEIF Lang$ = "GERMAN" THEN
       ReadGermanText Doc.First, Doc.Last, TextEMS%, NumLines%
END IF
```

```
'------------------------------------------ Parse Text into Words
' NumWords = number of words parsed

WordParse TextEMS%, NumLines, WordEMS%, NumWords&
EmsRelMem TextEMS%' release handle of DOC file, not needed anymore

'------------------------------------------ Find Combined Keys
' 50% ratio for expected combined keys

' 8192 keys per page (16384 page size/2 keycode size)

CombRatio! = .5
DispStat MID$(STR$(CLNG(NumWords& * CombRatio!)), 2) + " Combined Keys allocated."

NumPages = NumWords& * CombRatio! \ 8192 + 1
EmsAlloc NumPages, CombCodeEMS%, "CombCodeEMS"
a11! = TIMER
FindCombKey WordEMS%, NumWords&, CombKeywordEMS%, NumCombKeyword, CombCodeEMS%, NumCombCode
a12! = a12! + TIMER - a11!
DispStat Num$(NumCombCode) + " combined keys found."

'---- Store the Combined Keywords in DocKeysEMS

DocNdx.Ndx = CurrDocKey&
Index& = CurrDocKey&

'if this is a first appearence then put code to DocKeyTemp,i.e. store it and
'increase number of documents contained this word (IDF) and increase string(we
'keep previous codes as string (\**\**\**...))
'if not add frequency this word in this particular documente
k = 0
FOR i = 1 TO NumCombCode
        EmsGetIEl CodeTemp, LENCode, i, CombCodeEMS%
        CodeStr$ = "\" + MKI$(CodeTemp.Code)
        CodePos = INSTR(CodeString$, CodeStr$)
        IF CodePos = 0 THEN
                k = k + 1
                Idf&(CodeTemp.Code) = Idf&(CodeTemp.Code) + 1
                DocKeyTemp(k).Code = CodeTemp.Code
                DocKeyTemp(k).Freq = 1
                '-- since it wasn't found, it's new so we'll put it at CurrDocKey&
                '   and increment CurrDocKey& to point to the next available slot
                CurrDocKey& = CurrDocKey& + 1
                CodeString$ = CodeString$ + CodeStr$

        ELSE
                CodePos = CodePos \ 3 + 1
                DocKeyTemp(CodePos).Freq = DocKeyTemp(CodePos).Freq + 1
        END IF
NEXT
EmsRelMem CombCodeEMS%

'QPrintRC STR$(CurrDocKey& - 1), 3, 70, -1

'------------------------------------------ Find Single Keywords

SingRatio! = 1.5  'we need more space for word with and without prefixes
DispStat LTRIM$(STR$(CLNG(NumWords& * SingRatio!))) + " Single Keys allocated."
NumPages = NumWords& * SingRatio! \ 8192 + 1 '-- 8192 keycodes per 16k EMS page
EmsAlloc NumPages, SingCodeEMS%, "SingleCodeEMS"
a13! = TIMER
FindSingKey WordEMS%, NumWords&, SingKeywordEMS%, NumKeyword, SingCodeEMS%, NumSingCode&
a14! = a14! + TIMER - a13!
DispStat LTRIM$(STR$(NumSingCode&)) + " single keys found."

'---- don't need the parsed word list anymore so release the EMS
EmsRelMem WordEMS%

'the same as for combined keywords:
'if this is a first appearence then put code to DocKeyTemp,i.e. store it and
'increase number of documents contained this word (IDF) and increase string(we
'keep previous codes as string (\**\**\**...))
'if not add frequency this word in this particular documente

k& = LEN(CodeString$) \ 3  'number of unique keywords
FOR i& = 1 TO NumSingCode&
        EmsGet CodeTemp, LENCode, i&, SingCodeEMS%
        CodeStr$ = "\" + MKI$(CodeTemp.Code)
        CodePos = INSTR(CodeString$, CodeStr$)
        IF CodePos = 0 THEN
                k& = k& + 1
                IF k& > MaxTempKey THEN REDIM PRESERVE DocKeyTemp(1 TO k&) AS DocKeyType
                Idf&(CodeTemp.Code) = Idf&(CodeTemp.Code) + 1
                DocKeyTemp(k&).Code = CodeTemp.Code
                DocKeyTemp(k&).Freq = 1
                '-- since it wasn't found, it's new so we'll put it at CurrDocKey&
                '   and increment CurrDocKey& to point to the next available slot
                CurrDocKey& = CurrDocKey& + 1
                CodeString$ = CodeString$ + CodeStr$
        ELSE
                CodePos = CodePos \ 3 + 1
                DocKeyTemp(CodePos).Freq = DocKeyTemp(CodePos).Freq + 1
        END IF
NEXT

'QPrintRC STR$(CurrDocKey& - 1), 3, 70, -1

DocNdx.Num = CurrDocKey& - Index&
DocNdx.Tot = NumSingCode& + NumCombCode

DispStat MID$(STR$(NumSingCode& + NumCombCode), 2) + " Keywords."
DispStat MID$(STR$(CurrDocKey& - Index&), 2) + " Unique Keywords."

'------------------------------------------
```

```
'release memory for the combined and single keywords in EMS
EmsRelMem SingCodeEMS%

'store DocKeyTemp Array on disk , there is not enough memory to keep it in RAM
'and there is no sense to put it to EMS and later read back to RAM & put to disk
k& = LEN(CodeString$) \ 3
FOR i& = 1 TO k&
        CALL FPutRT(DocKeyFile%, DocKeyTemp(i&), Index& + i& - 1, LENDocKeyTemp)
NEXT
CALL FPutRT(DocNdxFileX, DocNdx, CLNG(File&), LenDocNdx) 'write ndx for each document

IF INKEY$ = CHR$(27) THEN
        chime 10
        DO: LOOP UNTIL INKEY$ = ""
        x$ = INPUT$(1)
        IF x$ = CHR$(13) THEN Proc = File&: File& = Count&
END IF
CodeString$ = ""
FreeStrSp& = FRE("")

'-- do an intermediate save every 100 docs
IF (File& MOD 100) = 0 THEN
        DispMsg "Intermediate Save at" + STR$(File&), r, c

'       '-- dump out the portion of the DocNdx() used (i.e., from Start to File)
'       FOR i& = NdxPoint TO File&
'               CALL EmsGet(SEG DocNdx, LenDocNdx, i&, DocNdxEMS%)
'               CALL FPutRT(DocNdxFileX, DocNdx, i&, LenDocNdx)
'       NEXT
'       NdxPoint = File& + 1
'       FPutAH LstDir$ + "DOCINDEX.AH" + Machine$, SEG DocNdx(Start&), LEN(DocNdx(Start&)), File& - Start& + 1
'
'       '-- save DocKeys from the last save point
'       FOR i& = AHSavePoint& TO CurrDocKey& - 1
'               CALL EmsGet(SEG DocKeyTemp, LENDocKeyTemp, i&, DocKeysEMS%)
'               CALL FPutRT(DocKeyFileX, DocKeyTemp, i&, LENDocKeyTemp)
'       NEXT
'       '-- set the save point to the next dockey
'       AHSavePoint& = CurrDocKey&

        '-- save the IDF.SIN
        CALL FPutAH(LstDir$ + "IDF" + Machine$ + ".SIN", SEG Idf&(1), -2, UBOUND(Idf&))

        DispMsg "", 0, 0

END IF
DispStat ""
NEXT File&
'*****************************************************************************
CLOSE #8, #9

IF Proc <> 0 THEN Count& = Proc
TEnd! = TIMER
PRINT #5, ""
PRINT #5, "End Time:"; TIME$; TEnd!; (TEnd! - TStart!) / Count&; "seconds per document."
CLOSE #5
TotalTime! = TEnd! - TStart!
IF TotalTime! < 0 THEN TotalTime! = TotalTime! + 86400!
PRINT TotalTime! / 60; "minutes elapsed time."
PRINT "FindCombKey"; a12!
PRINT "FindSingKey"; a14!
'=============================================================================

'release memory for COMBKEY and SINGKEY lists and the DICT.COD list

IF SingKeywordEMS% THEN EmsRelMem SingKeywordEMS%
IF Comb1KeywordEMS% THEN EmsRelMem Comb1KeywordEMS%
IF Comb2KeywordEMS% THEN EmsRelMem Comb2KeywordEMS%
IF Comb3KeywordEMS% THEN EmsRelMem Comb3KeywordEMS%
IF Sing1KeywordEMS% THEN EmsRelMem Sing1KeywordEMS%
IF Sing2KeywordEMS% THEN EmsRelMem Sing2KeywordEMS%

'decrement current pointer so that it points to the end of the array
'not at the next available space
CurrDocKey& = CurrDocKey& - 1

'---- Save storage to disk
IF NOT Debug THEN

        DispMsg "Saving IDF" + Machine$ + ".SIN", r, c
        FPutAH LstDir$ + "Idf" + Machine$ + ".SIN", SEG Idf&(1), -2, UBOUND(Idf&)
        DispMsg "", 0, 0

'       DispMsg "Saving DocIndex.AH" + Machine$, r, c
'       FOR i& = NdxPoint TO File&
'               CALL EmsGet(SEG DocNdx, LenDocNdx, i&, DocNdxEMS%)
'               CALL FPutRT(DocNdxFileX, DocNdx, i&, LenDocNdx)
'       NEXT
'
'       FPutAH LstDir$ + "DOCINDEX.AH" + Machine$, SEG DocNdx(Start&), LEN(DocNdx(Start&)), Count& - Start& + 1
'       DispMsg "", 0, 0
'
'       DispMsg "Saving DocKeys.AH" + Machine$ + " from EMS", r, c
'
'       FOR i& = AHSavePoint& TO CurrDocKey&
'               EmsGet SEG DocKeyTemp, LENDocKeyTemp, i&, DocKeysEMS%
'               FPutRT DocKeyFileX, DocKeyTemp, i&, LENDocKeyTemp
'       NEXT

        FClose DocKeyFile%
        FClose DocNdxFile%
END IF
```

```
'EmsRelMem DocKeysEMS%   '— release memory for Doc Keys
DispMsg "", 0, 0
chime 10
'$Page
END

SUB CheckUCaseWord (Word$, PrevWord$, Length, NoNameFlag) STATIC

NoNameFlag = FALSE

FOR i = 1 TO 4
        IF INSTR(NameEnding$(i), RIGHT$(Word$, i)) THEN
                IF Length > 9 THEN
                        NoNameFlag = TRUE
                        EXIT SUB
                END IF
        END IF
NEXT

IF Length > 7 THEN
        NoNameFlag = TRUE
        EXIT SUB
END IF

IF INSTR("\a//\an/\any\no/\this\such\many\several\", "\" + PrevWord$ + "\") THEN
        IF Length > 3 THEN
                NoNameFlag = TRUE
                EXIT SUB
        END IF
END IF

IF LCASE$(PrevWord$) = "the" AND Length > 4 THEN
        NoNameFlag = TRUE
        EXIT SUB
END IF

FOR i = 2 TO 6
        IF INSTR(NoNameEnding$(i), RIGHT$(Word$, i)) THEN
                IF Length > 2 THEN
                        NoNameFlag = TRUE
                        EXIT SUB
                END IF
        END IF
NEXT

END SUB

SUB Config (Machine$, First&, Last&) STATIC

IF NOT EmsLoaded% THEN
        CLS
        PRINT "AIM Fatal Error:"
        PRINT
        PRINT "No EMS drive was found."
        PRINT
        chime 10
        PRINT "Press the SPACE BAR to exit:"
        i$ = INPUT$(1)
        END
END IF

LENCode = LEN(CodeTemp)
SixteenK = 16 * 1024
SixtyFour = 64
ThirtyTwo = 32
ThirtyTwoK& = ThirtyTwo * 1024&
ASCEND = 0
DESCEND = NOT ASCEND
FALSE = 0
TRUE = NOT FALSE
ENTER = 13
ESCAPE = 27
ASCa = ASC("a")
ASCz = ASC("z")
ASCupperA = ASC("A")
ASCupperZ = ASC("Z")
ASCslash = ASC("/")
ASC0 = ASC("0")
ASC9 = ASC("9")
ASCat = ASC("@")

Cmd$ = QPTrim$(COMMAND$)

Parms = InCount(Cmd$, " ") + 1   '— number of parameters

IF Parms = 4 THEN

        '— Expected information on command line:
        '   Config file, Machine #, First Doc, Last Doc

        Extract Cmd$, " ", 1, Strt, SLen '— extract first parm
        ConfigFile$ = MID$(Cmd$, Strt, SLen) + ".CFG"

        Extract Cmd$, " ", 2, Strt, SLen '— extract second parm
        Machine$ = MID$(Cmd$, Strt, SLen)

        Extract Cmd$, " ", 3, Strt, SLen '— extract third parm
        First& = VAL(MID$(Cmd$, Strt, SLen))

        Extract Cmd$, " ", 4, Strt, SLen '— extract fourth parm
        Last& = VAL(MID$(Cmd$, Strt, SLen))
```

```
ELSE

        PRINT
        PRINT "AIM Program Error: Missing Parameters"
        PRINT
        PRINT
        PRINT "Required Parameters are:"
        PRINT
        PRINT "AIM  Config File  Machine Number  First Doc  Last Doc"
        PRINT
        chime 10
        PRINT "Press the SPACE BAR to exit:"
        i$ = INPUT$(1)
        END

END IF

IF NOT Exist(ConfigFile$) THEN
        chime 10
        PRINT "File "; ConfigFile$; " was not found."
        PRINT "Press any key to return to the system."
        DO: ch$ = INKEY$: LOOP UNTIL LEN(ch$) <> 0
        END
END IF
OPEN ConfigFile$ FOR INPUT ACCESS READ SHARED AS #1
        INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
CLOSE #1

COLOR Fg, Bg, Brdr

NormAttr = OneColorX(Fg, Bg)
RevAttr = OneColorX(Bg, Fg AND 7)

File$ = LstDir$ + "a" + Lang$ + ".LST"
IF NOT Exist(File$) THEN
        chime 10
        PRINT "File "; File$; " was not found."
        PRINT "Press any key to return to the system."
        DO: ch$ = INKEY$: LOOP UNTIL LEN(ch$) <> 0
        END
END IF
OPEN File$ FOR INPUT ACCESS READ SHARED AS 1
        INPUT #1, AtList$
CLOSE #1

File$ = LstDir$ + Lang$ + ".SEC"
IF NOT Exist(File$) THEN
        chime 10
        PRINT "File "; File$; " was not found."
        PRINT "Press any key to return to the system."
        DO: ch$ = INKEY$: LOOP UNTIL LEN(ch$) <> 0
        END
END IF
OPEN File$ FOR INPUT ACCESS READ SHARED AS 1
        INPUT #1, SecCode, ArtCode
CLOSE

RESTORE NLGData
REDIM NLG$(1 TO 8), LENNLG(1 TO 8)
FOR i = 1 TO 8
        READ NLG$(i)
        LENNLG(i) = LEN(NLG$(i))
NEXT

RESTORE EnglishData
REDIM Section$(1 TO 3)
FOR i = 1 TO 3
        READ Section$(i)
NEXT

REDIM Article$(1 TO 3)
FOR i = 1 TO 3
        READ Article$(i)
NEXT

RESTORE GermanData
REDIM Paragraph$(1 TO 4)
FOR i = 1 TO 4
        READ Paragraph$(i)
NEXT

REDIM Artikel$(1 TO 3)
FOR i = 1 TO 3
        READ Artikel$(i)
NEXT

RESTORE NumData
REDIM Numbers$(1 TO 30)
FOR i = 1 TO 30
        READ Numbers$(i)
NEXT

IF Lang$ <> "GERMAN" THEN
        REDIM NameEnding$(1 TO 4)
        REDIM NoNameEnding$(2 TO 6)
        RESTORE NameEnd
        FOR i = 1 TO 4
                READ NameEnding$(i)
        NEXT
        RESTORE NoNameEnd
```

```
        FOR i = 2 TO 6
                READ NoNameEnding$(i)
        NEXT
END IF
IF Lang$ = "GERMAN" THEN
        RESTORE GermanPrefixes
ELSE
        RESTORE EnglishPrefixes
END IF
REDIM Prefixes$(2 TO 9)
IF Lang$ = "GERMAN" THEN
        FOR i = 2 TO 9
                READ FirstHalf$, SecondHalf$
                Prefixes$(i) = FirstHalf$ + SecondHalf$
        NEXT
ELSE
        FOR i = 2 TO 9
                READ Prefixes$(i)
        NEXT
END IF

REDIM MeanPrefixes$(3 TO 14)
IF Lang$ = "GERMAN" THEN
        FOR i = 3 TO 14
                READ FirstHalf$, SecondHalf$, ThirdHalf$
                MeanPrefixes$(i) = FirstHalf$ + SecondHalf$ + ThirdHalf$
        NEXT
ELSE
        FOR i = 3 TO 14
                READ MeanPrefixes$(i)
        NEXT
END IF
END SUB

SUB DispMsg (Msg$, r, c) STATIC

STATIC WindOpen, Scr%() ' is there already a message displayed?

SHARED fg, Bg

IF Msg$ = "" THEN
        IF WindOpen THEN GOSUB MsgClose
        EXIT SUB
END IF

IF WindOpen THEN
        CALL chime(9)
        OPEN "DEBUG" FOR OUTPUT AS 10
        PRINT #10, "WindOpen="; WindOpen; HEX$(WindOpen)
        PRINT #10, "Msg$= |"; Msg$; "|"
        PRINT #10, "TRUE="; TRUE; HEX$(TRUE)
        PRINT #10, "FALSE="; FALSE; HEX$(FALSE)
        CLOSE 10
        CALL chime(8)
        CLS
        END
        i$ = INPUT$(1)
        GOSUB MsgClose
END IF

Wid = LEN(Msg$)
IF Wid > 50 THEN Wid = 50

Msg$ = Msg$ + " "' make sure there's a space to find at the end (see below)

MaxLin = LEN(Msg$) \ Wid + 3
IF MaxLin > 23 THEN MaxLin = 23
REDIM Text$(MaxLin)
Lin = 0
DO
        Lin = Lin + 1' incr current lin # (also element in text display array)
        Lastspc = QInstrRC%(Wid + 1, Msg$, " ")' look for the last space so we can word wrap
        Text$(Lin) = LEFT$(Msg$, Lastspc - 1)
        Msg$ = MID$(Msg$, Lastspc + 1)'remove portion of string that's in t$
LOOP WHILE LEN(Msg$) > Wid

Msg$ = RTRIM$(Msg$)
IF LEN(Msg$) THEN
        Lin = Lin + 1
        Text$(Lin) = Msg$
END IF

vertmargin = (25 - Lin) / 2
IF r <> 0 AND c = 0 THEN
        ULr = r
ELSE
        ULr = 9'ulr = INT(vertmargin - .5)
END IF
  DULr = ULr - 1
  LRr = ULr + Lin + 1'LRr = 25 - INT(vertmargin)
  DLRr = LRr + 2

horizmargin = (80 - Wid) \ 2
  ULc = horizmargin
  DULc = ULc - 3
  LRc = 80 - INT(horizmargin)
  IF Wid / 2 = Wid \ 2 THEN LRc = LRc + 1
  DLRc = LRc + 1

REDIM Scr%(ArraySize%(DULr, DULc, DLRr, DLRc))
CALL ScrnSaveO(DULr, DULc, DLRr, DLRc, SEG Scr%(0))
CALL WindMgr(ULr, ULc, LRr, LRc, 4, NormAttr, RevAttr, "Status")
FOR i = 1 TO Lin
        CALL QPrintRC(Text$(i), ULr + i, ULc + 1, -1)
```

```
NEXT

r = ULr + Lin
c = ULc + 1 + LEN(Text$(Lin))
IF LEN(Text$(Lin)) + 2 = Wid THEN c = ULc + 1: r = r + 1

ERASE Text$
WindOpen = TRUE

EXIT SUB

'------------------------------------------------------ Close Window
MsgClose:

CALL ScrnRest0(DULr, DULc, DLRr, DLRc, SEG Scr%(0))
ERASE Scr%
WindOpen = FALSE
RETURN

END SUB

SUB DispStat (s$) STATIC

STATIC Row

IF Row = 0 THEN
        Row = 15
        Col = 23
        Height = 8
        Wid = 34
        CALL WindMgr(Row, Col - 1, Row + Height, Col + Wid, 1, NormAttr, RevAttr, "FILE STATISTICS")
        Col = Col + 2
ELSE
        Row = Row + 1
END IF

IF s$ = "" THEN
        Row = 0
ELSE
        CALL QPrintRC(s$, Row, Col, -1)
END IF

IF INSTR(s$, "alloc") = 0 THEN PRINT #5, s$

END SUB

SUB EmsAlloc (NumPages%, Handle%, LoadFILE$) STATIC

EmsAllocMem NumPages%, Handle%
IF EmsError% THEN
        PRINT "Couldn't allocate"; CLNG(NumPages) * Sixteen%; "bytes of EMS for "; LoadFILE$
        chime 2
        DO: LOOP UNTIL LEN(INKEY$) = 0
        i$ = INPUT$(1)
        END
END IF

END SUB

SUB FindCombKey (WordEMS%, NumWords&, KeyEMS%, NumKey%, CombFoundEMS%, NumCombFound%) STATIC

'EMS ARRAY  LEN DESCRIPTION              DIRECTION   MODIFIED?
'---------- --- -------------------------- ---------   ---------
'WordH:      32 Document Words          (Passed)    (Unchanged)
'KeyH:       64 Combined Keywords       (Passed)    (Unchanged)
'CombFoundH: 64 Combined Keywords Found (Returned)  (Changed)

DIM WordTemp AS STR32      '-- current document word
DIM WordCompare AS STR32   '-- current word in CombKey to check
DIM KeyTemp AS CombKeyType '-- entire Combined Keyword
IF Lang$ = "GERMAN" THEN
        REDIM Comb1ArrayGer(1 TO NumComb1Keyword)  AS CombKeyType
        REDIM Comb2ArrayGer(1 TO NumComb2Keyword)  AS CombKeyType
        REDIM Comb3ArrayGer(1 TO NumComb3Keyword)  AS CombKeyType
        EMS2ARRAY Comb1ArrayGer(1), LenComb, NumComb1Keyword, Comb1KeywordEMS%
        EMS2ARRAY Comb2ArrayGer(1), LenComb, NumComb2Keyword, Comb2KeywordEMS%
        EMS2ARRAY Comb3ArrayGer(1), LenComb, NumComb3Keyword, Comb3KeywordEMS%
ELSE
        REDIM Comb1Array(1 TO NumComb1Keyword)  AS NewCombType
        REDIM Comb2Array(1 TO NumComb2Keyword)  AS NewCombType
        REDIM Comb3Array(1 TO NumComb3Keyword)  AS NewCombType
        EMS2ARRAY Comb1Array(1), LenComb, NumComb1Keyword, Comb1KeywordEMS%
        EMS2ARRAY Comb2Array(1), LenComb, NumComb2Keyword, Comb2KeywordEMS%
        EMS2ARRAY Comb3Array(1), LenComb, NumComb3Keyword, Comb3KeywordEMS%
END IF
NumCombFound = 0
SecondEnd = NumComb2Keyword + NumComb1Keyword
LenWord = LEN(WordTemp)
LENKey = LEN(KeyTemp)

Slash$ = "///"
ASCslash = ASC("/")

IF Lang$ = "ENGLISH" THEN English = TRUE ELSE English = FALSE

d$ = "Marking Combined Keywords: x     out of" + STR$(NumWords&)
x = LEN(d$) - INSTR(d$, "x")
DispMsg d$, r, c
c = c - x - 1

FOR i& = 1 TO NumWords&' number of words in document
```

```
QPrintRC MID$(STR$(i8), 2), r, c, -1

'-- get word from list of parsed document words
EmsGet WordTemp, LenWord, i8, WordEMS
'-- convert it to a variable-length for speed
WordTempStr$ = RTRIM$(WordTemp.Str)

'-- if it's English, then make it Lowercase since we ignore case
'   for Combined Keywords
IF English THEN
        Lower WordTempStr$
        IF RIGHT$(WordTempStr$, 1) = CHR$(255) THEN
                WordTempStr$ = LEFT$(WordTempStr$, LEN(WordTempStr$) - 1)
        END IF
END IF
'-- if it's a valid range, then check words in range
IF FirstLastX(WordTempStr$, First, Last, Comb) THEN

        FOR j = Last TO First STEP -1

          '-- get word from Combined Keyword List (COMBKEY.STR)
          IF Lang$ = "GERMAN" THEN
                IF j <= NumComb1Keyword THEN
                        KeyTempStr$ = RTRIM$(Comb1ArrayGer(j).Str)
                ELSEIF j <= SecondEnd THEN
                        KeyTempStr$ = RTRIM$(Comb2ArrayGer(j - NumComb1Keyword).Str)
                ELSE
                        KeyTempStr$ = RTRIM$(Comb3ArrayGer(j - SecondEnd).Str)
                END IF
          ELSE
                IF j <= NumComb1Keyword THEN
                        KeyTempStr$ = RTRIM$(Comb1Array(j).Str)
                ELSEIF j <= SecondEnd THEN
                        KeyTempStr$ = RTRIM$(Comb2Array(j - NumComb1Keyword).Str)
          ELSE
                KeyTempStr$ = RTRIM$(Comb3Array(j - SecondEnd).Str)
          END IF
END IF
Words = InCount(KeyTempStr$, " ") + 1'count number of words
      CALL Extract(KeyTempStr$, " ", 1, Strt, SLen)'extract first word
      CurrKey$ = MID$(KeyTempStr$, Strt, SLen)'     of combined keyword

      IF MidCharX(CurrKey$, SLen) = ASCslash THEN
              Exact = TRUE
              CurrKey$ = LEFT$(CurrKey$, SLen - 1)
              SLen = SLen - 1
      ELSE
              Exact = FALSE
      END IF

      IF SLen < 3 THEN
              CurrKey$ = CurrKey$ + LEFT$(Slash$, 3 - SLen)
              SLen = 3
      END IF

      'compare first word of combined key [CurrKey$]
      'against the current document word [WordTempStr$]

      IF English THEN
              IF NOT Exact THEN
                      Match = (LCASE$(CurrKey$) = LEFT$(WordTempStr$, SLen))
              ELSE ' check for "exact" match
                      Match = (LCASE$(CurrKey$) = WordTempStr$)
              END IF
      ELSE '-- German
              IF NOT Exact THEN
                      Match = (CurrKey$ = LEFT$(WordTempStr$, SLen))
              ELSE ' check for "exact" match
                      Match = (CurrKey$ = WordTempStr$)
              END IF
      END IF

      ' no match, skip to next combined key in the First-Last range
      IF NOT Match GOTO SkipCombKey

      ' continue matching the rest of the words in the combined key
      ' exiting out as soon as there's a non-match

      AtFlag = FALSE
      NotFlag = FALSE

      FOR k = 2 TO Words' number of words left in combined key

              ' extract the next word from the current combined keyword (j)
              CALL Extract(KeyTempStr$, " ", k, Strt, SLen)
              CurrKey$ = MID$(KeyTempStr$, Strt, SLen)

              IF MidCharX(CurrKey$, SLen) = ASCslash THEN
                      Exact = TRUE
                      CurrKey$ = LEFT$(CurrKey$, SLen - 1)
                      SLen = SLen - 1
              ELSE
                      Exact = FALSE
              END IF

              IF SLen < 3 THEN
                      CurrKey$ = CurrKey$ + LEFT$(Slash$, 3 - SLen)
                      SLen = 3
              END IF

              IF AtFlag = FALSE AND NotFlag = FALSE THEN
                      EmsGet WordCompare, LenWord, i8 + k - 1, WordEMS
```

```
ELSE
        IF AtFlag = FALSE AND NotFlag = TRUE THEN
                EmsGet WordCompare, LenWord, i& + k, WordEMS%
        ELSE
                IF AtFlag = TRUE AND NotFlag = FALSE THEN
                        EmsGet WordCompare, LenWord, i& + k - 2, WordEMS%
                ELSE
                        EmsGet WordCompare, LenWord, i& + k - 1, WordEMS%
                END IF
        END IF
END IF
DocWord$ = RTRIM$(WordCompare.Str)' Document word to compare

IF English THEN Lower DocWord$

IF ASCII(CurrKey$) <> ASCat THEN
        IF English THEN
                IF Exact THEN ' check for "exact" match
                        Match = (LCASE$(CurrKey$) = DocWord$)
                ELSE ' wildcard match, only compare # of chars in CurrKey$
                        Match = (LCASE$(CurrKey$) = LEFT$(DocWord$, SLen))
                END IF
        ELSE '-- German: no need to use Lcase$
                IF Exact THEN ' check for "exact" match
                        Match = (CurrKey$ = DocWord$)
                ELSE ' wildcard match, only compare # of chars in CurrKey$
                        Match = (CurrKey$ = LEFT$(DocWord$, SLen))
                END IF
        END IF
ELSE '-- special processing for a wildcard
        IF INSTR(AtList$, "/" + DocWord$ + "/") THEN
                Match = TRUE' the word was in the @ list, so continue
        ELSE
                IF English THEN
                        Match = FALSE
                ELSE
                        Match = TRUE
                        AtFlag = TRUE
                END IF
        END IF
        IF Match THEN
                EmsGet WordCompare, LenWord, i& + k, WordEMS%
                DocWord$ = RTRIM$(WordCompare.Str)
                IF DocWord$ = "not" OR DocWord$ = "be" OR DocWord$ = "nicht" THEN NotFlag = TRUE
        END IF
END IF

IF NOT Match GOTO SkipCombKey

NEXT' word in current combined keyword

IF Match THEN ' this is a combined keyword, so add it to the list
        NumCombFound = NumCombFound + 1
        IF Lang$ = "GERMAN" THEN
                IF j <= NumComb1Keyword THEN
                 CodeTemp.Code = Comb1ArrayGer(j).Code
                ELSEIF j <= SecondEnd THEN
                 CodeTemp.Code = Comb2ArrayGer(j - NumComb1Keyword).Code
                ELSE
                 CodeTemp.Code = Comb3ArrayGer(j - SecondEnd).Code
                END IF
        ELSE
                IF j <= NumComb1Keyword THEN
                 CodeTemp.Code = Comb1Array(j).Code
                ELSEIF j <= SecondEnd THEN
                 CodeTemp.Code = Comb2Array(j - NumComb1Keyword).Code
                ELSE
                 CodeTemp.Code = Comb3Array(j - SecondEnd).Code
                END IF
        END IF
        EmsSet1EL CodeTemp, LENCode, NumCombFound, CombFoundEMS%
        IF EmsErrorX THEN '-- probably ran out of storage in EMS
                NumCombFound = NumCombFound - 1
        END IF

        EXIT FOR
END IF

SkipCombKey:

        NEXT
    END IF' Table range was valid

    IF EmsError% THEN EXIT FOR

NEXT' key in list

DispMsg "", r, c
IF Lang$ = "GERMAN" THEN
        ERASE Comb1ArrayGer
        ERASE Comb2ArrayGer
        ERASE Comb2ArrayGer
ELSE
        ERASE Comb1Array
        ERASE Comb2Array
        ERASE Comb2Array
END IF
END SUB

SUB FindSingKey (WordEMS%, NumWords&, KeyEMS%, NumKey%, SingFoundEMS%, NumSingFound$) STATIC

'EMS ARRAY  LEN DESCRIPTION          * DIRECTION   MODIFIED?
'----------  --- -------------------- ----------   ---------
```

```
'WordH:      32 Document Words          (Passed)    (Unchanged)
'KeyH:       32 Single Keywords         (Passed)    (Unchanged)
'SingFoundH: 32 Single Keywords Found   (Returned)  (Changed)

DIM WordTemp AS STR32       '— current document word
DIM KeyTemp AS SingKeyType '— Single Keyword to be compared
LenWord = LEN(WordTemp)
LENKey = LEN(KeyTemp)

ASCslash = ASC("/")

NumSingFound& = 0

d$ = "Marking Single Keywords: x     out of" + STR$(NumWords&)
x = LEN(d$) - INSTR(d$, "x")
DispMsg d$, r, c
c = c - x - 1

FOR i& = 1 TO NumWords&' number of words in document

        QPrintRC LTRIM$(STR$(i&)), r, c, -1
        EmsGet WordTemp, LenWord, i&, WordEMS%
        PrefixFlag = FALSE: MeanPrefixFlag = FALSE: UpperCaseFlag = FALSE
        Limit = 0
        WordTempStr$ = RTRIM$(WordTemp.Str)
        IF RIGHT$(WordTempStr$, 1) = CHR$(255) THEN
                WordTempStr$ = LEFT$(WordTempStr$, LEN(WordTempStr$) - 1)
                NewSentFlag = TRUE
        ELSE
                NewSentFlag = FALSE
        END IF

        ' check if the first 3 letters of the word return
        ' a valid range from the 3-dimensional table array
TryAgain:
        IF FirstLastX(LCASE$(WordTempStr$), First, Last, Sing) THEN  ' yes, so search thru range

                FOR j = Last TO First STEP -1

                        '— get the word from the SINGKEY.STR list
                        EmsGetIEl KeyTemp, LENKey, j, KeyEMS%
                        CurrKey$ = RTRIM$(KeyTemp.Str)
                        SLen = LEN(CurrKey$)

                        '— compare the single keyword [CurrKey$/KeyTemp.Str]
                        '   against the document word [WordTemp.Str]
                        '— [replaced] IF RIGHT$(CurrKey$, 1) = "/" THEN

                        IF MidCharX(CurrKey$, SLen) = ASCslash THEN
                                CurrKey$ = LEFT$(CurrKey$, SLen - 1)
                                Match = (CurrKey$ = RTRIM$(WordTempStr$))
                        ELSE
                                Match = (CurrKey$ = LEFT$(WordTempStr$, SLen))
                        END IF

                        IF Match AND LEN(CurrKey$) >= Limit THEN   '— add the single keyword to the list
                                IF NOT UpperCaseFlag THEN
                                        NumSingFound& = NumSingFound& + 1
                                        CodeTemp.Code = KeyTemp.Code
                                        EmsSet CodeTemp, LENCode, NumSingFound&, SingFoundEMS%

                                        IF EmsError% THEN '— probably ran out of storage in EMS
                                                NumSingFound = NumSingFound - 1
                                        END IF
                                END IF
                                EXIT FOR
                        END IF
                NEXT' key in range

        ELSE 'check for sections
        Letter$ = LEFT$(WordTemp.Str, 2)
        IF Letter$ = "zs" OR Letter$ = "za" THEN
                SecNum& = VAL(MID$(WordTemp.Str, 3))
                IF SecNum& <= 3000 THEN
                        IF SecNum& THEN
                                IF Letter$ = "za" THEN
                                        CodeTemp.Code = SecNum& + SecCode        '10563
                                ELSE
                                        CodeTemp.Code = SecNum& + ArtCode        '13563
                                END IF
                                IF CodeTemp.Code <= ArtCode + 30 THEN
                                        NumSingFound& = NumSingFound& + 1
                                        EmsSet CodeTemp, LENCode, NumSingFound&, SingFoundEMS%
                                        IF EmsError% THEN '— probably ran out of storage in EMS
                                                NumSingFound = NumSingFound - 1
                                        END IF
                                        Match = TRUE
                                ELSE  '— there's an error
                                        ErrorFILE% = FREEFILE
                                        OPEN "error.txt" FOR APPEND AS ErrorFILE%
                                                PRINT #ErrorFILE%, "Document #:"; File&
                                                PRINT #ErrorFILE%, "A Code was out of range:"; CodeTemp.Code
                                                PRINT #ErrorFILE%, "WordTemp.Str='"; WordTemp.Str; "'"
                                                PRINT #ErrorFILE%, "SecNum="; SecNum&
                                                PRINT #ErrorFILE%, ""
                                        CLOSE ErrorFILE%
                                END IF
                        END IF   '— there was a number following zs or za
                END IF   '— if the secnum <=3000
        END IF   '— this was a zs (section) or za (section) word
```

```
END IF' the range was valid

'for the first word try upper-case
IF (NOT Match OR UpperCaseFlag) AND Lang$ <> "GERMAN" THEN
        IF NewSentFlag THEN 'try upper-case
                WordTempStr$ = UCASE$(LEFT$(WordTempStr$, 1)) + MID$(WordTempStr$, 2)
                NewSentFlag = FALSE
                GOTO TryAgain
        ELSE
                IF UpperCaseFlag THEN  'the word was changed already, check can we store it
                        IF Match THEN
                                EmsGet WordTemp, LenWord, 18 - 1, WordEMS%
                                PrevWord$ = RTRIM$(WordTemp.Str)
                                CALL CheckUCaseWord(WordTempStr$, PrevWord$, LEN(CurrKey$), NoNameFlag)
                                IF NoNameFlag THEN 'can store the word
                                        NumSingFound& = NumSingFound& + 1
                                        CodeTemp.Code = KeyTemp.Code
                                        EmsSet CodeTemp, LENCode, NumSingFound&, SingFoundEMS%
                                END IF
                        END IF
                        UpperCaseFlag = FALSE
                ELSE
                        'if it is upper-case word check it as a low-case only if it matches > 6 letters
                        FirstLet = ASC(LEFT$(WordTempStr$, 1))
                        IF FirstLet >= 65 AND FirstLet <= 90 THEN
                                MID$(WordTempStr$, 1, 1) = LCASE$(LEFT$(WordTempStr$, 1))
                                UpperCaseFlag = TRUE
                                GOTO TryAgain
                        END IF
                END IF
        END IF
END IF

IF NOT MeanPrefixFlag THEN
        'check for meaningful prefixes. If found, divide word in two parts
        WordTempStr$ = LCASE$(WordTempStr$)
        LenW = LEN(WordTempStr$)
        FOR NumLet = 14 TO 3 STEP -1
                IF LenW > NumLet + 3 THEN  'should leave at least 3 letters
                        IF INSTR(MeanPrefixes$(NumLet), "\" + LEFT$(WordTempStr$, NumLet) + "\") THEN
                                WordTemp1$ = MID$(WordTempStr$, NumLet + 1)
                                WordTempStr$ = LEFT$(WordTempStr$, NumLet)
                                MeanPrefixFlag = TRUE
                                PrevMatch = Match 'save, because Match will change for the pref.
                                EXIT FOR
                        END IF
                END IF
        NEXT
        IF MeanPrefixFlag THEN GOTO TryAgain 'check again
ELSE
                IF WordTemp1$ <> "" THEN
                        IF PrevMatch THEN
                                Limit = 9
                        ELSE
                                Limit = 6
                        END IF
                                WordTempStr$ = WordTemp1$
                                WordTemp1$ = ""
                                IF LEN(WordTempStr$) >= Limit THEN GOTO TryAgain
                END IF
        END IF

        'check for meaningless prefixes and delete it IF LEN(WordTempStr$) >= 6
        IF NOT PrefixFlag AND NOT Match THEN  'only one time
                WordTempStr$ = LCASE$(WordTempStr$)
                LenW = LEN(WordTempStr$)
                FOR NumLet = 9 TO 2 STEP -1
                 IF LenW > NumLet + 3 THEN  'should leave at least 3 letters
                        IF INSTR(Prefixes$(NumLet), "\" + LEFT$(WordTempStr$, NumLet) + "\") THEN
                                WordTempStr$ = MID$(WordTempStr$, NumLet + 1)
                                PrefixFlag = TRUE
                                EXIT FOR
                        END IF
                 END IF
                NEXT
                IF PrefixFlag THEN
                        Limit = 6
                        IF LEN(WordTempStr$) >= Limit THEN GOTO TryAgain
                END IF
        END IF
        IF EmsError% THEN EXIT FOR

'NEXT' word in document

DispMsg "", 0, 0

END SUB

FUNCTION FirstLast% (Word$, First%, Last%, KeyType%) STATIC

'-- returns the starting (First) and ending (Last) range for the word
'   by looking it up in the Table%() array

SHARED SingTable%(), CombTable%()

a = XLateTable%(ASC(Word$))
b = XLateTable%(MidChar(Word$, 2))
c = XLateTable%(MidChar(Word$, 3))
IF a = 0 OR b = 0 OR c = 0 THEN FirstLast% = 0: EXIT FUNCTION
```

```
IF KeyType% = Sing THEN
        First% = SingTable%(a, b, c, 1)
        Last% = SingTable%(a, b, c, 2)
ELSE
        First% = CombTable%(a, b, c, 1)
        Last% = CombTable%(a, b, c, 2)
END IF

'-- Return FALSE if there was no valid range (i.e., First%=0)
FirstLast% = (First% <> 0)

END FUNCTION

FUNCTION LoadIntoEMS (File$) STATIC

'--------- Returns the handle where the file was loaded into ---------

EMSPg = EmsGetPFSeg%
SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, FileEMS, File$

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
FOpenAll File$, 0, 4, LoadFILE
FOR i = 1 TO Num32kBlocks + 1

        BoxO 14, 10, 18, 70, 2, RevAttr
        PaintBoxO 14, 10, 18, 70, RevAttr
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1); - " ", 16, 12, RevAttr
         '-- map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT

        '-- seek to beginning of current block
        FSeek LoadFILE, (i - 1) * ThirtyTwoK&

        IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        IF i < Num32kBlocks + 1 THEN

                '-- get the 32k block and put it directly into the EMS page frame
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        ELSE

                '-- load the left over (<32k) bytes
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, LeftOver&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        END IF
NEXT

FClose LoadFILE

ClearScrO 14, 10, 18, 70, NormAttr

LoadIntoEMS = FileEMS

END FUNCTION

FUNCTION Num$ (x) STATIC

Num$ = MID$(STR$(x), 2)

END FUNCTION

SUB ReadEnglishText (First&, Last&, Handle%, Lin%) STATIC

DIM Temp AS STR80' for use in storing in EMS

LENTemp = LEN(Temp)

DispMsg "Loading file into memory, line #:  .", r, c

NumLines = 0   ' total number of lines input from file
ActualLin = 0  ' total number of lines minus any blank lines

EndOfSentence$ = ".!?"

FOR i& = First& TO Last&
        GET #9, i&, Temp.Str
        '-- replace all quotes with spaces so as not to complicate the
        '   lower-casing of the first words of sentences
        ReplaceChar Temp.Str, CHR$(34), " "
        ReadSection Temp.Str, Section$(), Article$()
        NumLines = NumLines + 1
        EmsSetIEL Temp, LENTemp, NumLines, Handle
NEXT

'---- Process text, first making all first letters of sentences lower case

CurrLine = 0
GOSUB GetNextLine
```

```
DO
        DO

                '— skip over blank lines, or if we've gone too far
                DO WHILE Start > LenTxt
                        IF CurrLine = NumLines GOTO EndOfFile
                        GOSUB GetNextLine
                LOOP
                p = InstrTblZX(Start, Txt$, EndOfSentence$)

                IF p = 0 THEN

                        IF CurrLine = NumLines GOTO EndOfFile
                        GOSUB GetNextLine
                        p = 0

                ELSE  '— check for a NLG (honorific/title)

                        Start = p + 2

                        FOR i = 1 TO UBOUND(NLG$)

                                IF LENNLG(i) < p THEN
                                        IF LCASE$(MID$(Txt$, p - LENNLG(i), LENNLG(i))) = NLG$(i) THEN

                                                p = 0
                                                EXIT FOR

                                        END IF
                                END IF

                        NEXT

                END IF

        LOOP UNTIL p

        p = p + 2
        IF p > LenTxt THEN
                IF CurrLine = NumLines THEN
                        GOTO EndOfFile
                ELSE
                        GOSUB GetNextLine
                END IF
        END IF

        CurrChar = MidCharX(Txt$, p)

        DO WHILE CurrChar < ASCupperA OR CurrChar > ASCupperZ

                p = p + 1
                IF p > LenTxt THEN
                        IF CurrLine = NumLines THEN
                                GOTO EndOfFile
                        ELSE
                                GOSUB GetNextLine
                        END IF
                END IF

                CurrChar = MidCharX(Txt$, p)

        LOOP

        MID$(Txt$, p, 1) = CHR$(CurrChar + 32)
        'add CHR$(255) to the end of word to indicate beginning of the sentences
        SpLoc = InstrTblL2(p, Txt$, " ,)}].!?")
        IF SpLoc <> 0 THEN MID$(Txt$, SpLoc, 1) = CHR$(255)
        Temp.Str = Txt$
        EmsSet1EL Temp, LENTemp, CurrLine, Handle
        Start = p + 1

EndOfFile:

LOOP UNTIL CurrLine >= NumLines

New$ = " "      ' replacement string for punctuation
Old$ = ":/.-()[]"
FOR j = 91 TO 96: Old$ = Old$ + CHR$(j): NEXT
LENOld = LEN(Old$)

FOR i = 1 TO NumLines

        EmsGet1El Temp, LENTemp, i, Handle

        Txt$ = RTRIM$(Temp.Str)

        CALL RemCtrl(Txt$, "")'       replace all ctrl chars with blanks

        ' replace only SOME punctuation with spaces
        FOR j = 1 TO LENOld
                CALL ReplaceChar(Txt$, MID$(Old$, j, 1), New$)
        NEXT

        CALL StripRange(Txt$, 33, 37, TLen)    '— strip Punctutation ! to %
        Txt$ = LEFT$(Txt$, TLen)
        CALL StripRange(Txt$, 39, 47, TLen)    '— strip ' to /   don't strip numbers
        Txt$ = LEFT$(Txt$, TLen)
        CALL StripRange(Txt$, 58, 64, TLen)    '— strip : to @
        Txt$ = LEFT$(Txt$, TLen)
        CALL StripRange(Txt$, 123, 254, TLen)  '— strip High chars
        Txt$ = LEFT$(Txt$, TLen)
```

```
        CALL Crunch(Txt$, " ", TLen)'   crunch all multiple spaces to 1
        Txt$ = LEFT$(Txt$, TLen)
        Txt$ = LTRIM$(RTRIM$(Txt$))'    remove spaces from left & right

        IF LEN(Txt$) THEN 'there's still text there, i.e. it wasn't all spaces
                ActualLin = ActualLin + 1
                Temp.Str = Txt$
                EmsSet1EL Temp, LENTemp, ActualLin, Handle
        END IF

        CALL QPrintRC(STR$(ActualLin), r, c - 4, -1)

NEXT
CLOSE #1

CALL DispMsg("", r, c)
CALL DispStat(Num$(ActualLin) + " lines in file.")

Lin = ActualLin'                    return the actual number of lines saved

EXIT SUB

'-------------------------------------------------------------

GetNextLine:

        CurrLine = CurrLine + 1
        Start = 1   '-- start scanning at first position
        p = 1

        EmsGet1EL Temp, LENTemp, CurrLine, Handle

        '-- trim down end of line, but make sure there's one space at the end
        '   so that we can find end of sentences (looking for a DOT & SPACE)
        '   even if they're at the end of the line.

        Txt$ = RTRIM$(Temp.Str) + " "
        LenTxt = LEN(Txt$)

RETURN

END SUB

SUB ReadGermanText (FirstLine&, LastLine&, Handle%, Lin%) STATIC

DIM Temp AS STR80' for use in storing in EMS
LENTemp = LEN(Temp)

CALL DispMsg("Loading file into memory, line #:    ", r, c)

ActualLin = 0' number of lines minus any blank lines

FOR i& = FirstLine& TO LastLine&

        GET #9, i&, Temp.Str
        Txt$ = QPRTrim$(Temp.Str)

        '-- first process the ReadSection for German
        ReadSection Txt$, Paragraph$(), Artikel$()

        Lower Txt$          '-- convert all chars to lower case
        RemCtrl Txt$, ""    '-- replace all ctrl chars with blanks

        '-- replace only SOME punctuation with spaces
        New$ = " "
        Old$ = ":/.-()[]{}"
        FOR j = 1 TO LEN(Old$)
                CALL ReplaceChar(Txt$, MID$(Old$, j, 1), New$)
        NEXT

        CALL StripRange(Txt$, 33, 37, TLen)' strip Punctutation ! to %
        Txt$ = LEFT$(Txt$, TLen)

        ' Note: the range is thru chr$(96) because all the letters are lower case
        '  and all numbers are being stripped out too.  We've skipped over 38
        '  because it's the & char which is allowed
        ' NOTE 2 [VN]: we no longer strip out numbers since we now use the section
        '  and article numbers

        CALL StripRange(Txt$, 39, 47, TLen)     '-- strip ' to /
        Txt$ = LEFT$(Txt$, TLen)
        CALL StripRange(Txt$, 58, 96, TLen)     '-- strip : to `
        Txt$ = LEFT$(Txt$, TLen)
        CALL StripRange(Txt$, 123, 255, TLen)' strip High chars
        Txt$ = LEFT$(Txt$, TLen)
        CALL Crunch(Txt$, " ", TLen)'-- crunch all multiple spaces to 1
        Txt$ = LEFT$(Txt$, TLen)
        Txt$ = QPTrim$(Txt$)        '-- remove spaces from left & right

        IF Txt$ <> "" THEN '        there's still text there, i.e. it wasn't all spaces
                ActualLin = ActualLin + 1
                Temp.Str = Txt$
                EmsSet1EL SEG Temp, LENTemp, ActualLin, Handle
        END IF

        QPrintRC STR$(ActualLin), r, c - 4, -1

NEXT
CLOSE #1
```

```
DispMsg "", r, c
DispStat Num$(ActualLin) + " lines in file."

Lin = ActualLin'                    return the actual number of lines saved

END SUB

SUB ReadSection (Txt$, SecArray$(), ArtArray$()) STATIC

'---Look for "sections" or "articles"

IF Lang$ = "GERMAN" THEN
        '-- In German it's Par
        SearchStr$ = "Par"
ELSE
        '-- In English it's Sec
        SearchStr$ = "Sec"
END IF

Letter$ = "zs"

FOR LookStep = 1 TO 2

        Start = 1

        DO
                m = INSTR(Start, Txt$, SearchStr$)  '-- column of start of Sec. or Art.

                IF m THEN

                        j = INSTR(m, Txt$, " ") '-- position of the end of the word

                        IF j THEN   '-- if this is not a last word
                                Word$ = MID$(Txt$, m, j - m)  '-- get the whole word
                        ELSE
                                EXIT DO  '-- this was the last word, so exit
                        END IF

                        '-- check if the word matches variations on Section or Article

                        IF Lang$ = "GERMAN" AND LookStep = 1 THEN
                                NumFound = 4
                        ELSE
                                NumFound = 3   '-- there are three variations that we check for
                        END IF

        IF LookStep = 1 THEN
                CALL FindExact(VARPTR(SecArray$(1)), NumFound, Word$)
        ELSE
                CALL FindExact(VARPTR(ArtArray$(1)), NumFound, Word$)
        END IF

        IF NumFound <> -1 THEN  '-- it did match, so check the number
                k = j + 1   '-- starting position of [potential] number

                DO '-- skip over blank spaces
                        ch = MidChar(Txt$, k)
                        k = k + 1
                LOOP UNTIL ch <> 32 OR k > LEN(Txt$)

                m1 = 0

                DO '-- collect the whole number
                        ch = MidChar(Txt$, k + m1 - 1)
                        m1 = m1 + 1
                        IF k + m1 - 1 > LEN(RTRIM$(Txt$)) THEN m1 = m1 + 1: EXIT DO
                LOOP UNTIL ch < ASC0 OR ch > ASC9

                IF m1 > 1 THEN 'there is a number
                        Numb$ = MID$(Txt$, k - 1, m1 - 1)
                        IF VAL(Numb$) <= 3000 AND VAL(Numb$) > 0 THEN
                                'if we picked up one character more because of end of line, delete it
                                IF RIGHT$(Numb$, 1) < "0" OR RIGHT$(Numb$, 1) > "9" THEN Numb$ = LEFT$(Numb$, LEN(Numb
                                '-- if we're looking for Article numbers, don't accept
                                '   article numbers over 30
                                IF LookStep = 2 AND VAL(Numb$) > 30 THEN GOTO NextStep

                                NewWord$ = Letter$ + Numb$
                                m1 = INSTR(k, Txt$, " ")
                                Txt1$ = LEFT$(Txt$, m - 1) + NewWord$
                                IF m1 <> 0 THEN
                                        Txt$ = Txt1$ + MID$(Txt$, m1)
                                ELSE
                                        Txt$ = Txt1$
                                END IF
                        END IF
                ELSE
                        IF SearchStr$ = "Art" THEN

                                m1 = 0

                                '-- loop while it's a Roman numeral and we're not
                                '   past the end of the string
                                DO
                                        ch$ = MID$(Txt$, k + m1 - 1, 1)
                                        m1 = m1 + 1
                                        IF k + m1 - 1 > LEN(RTRIM$(Txt$)) THEN EXIT DO
                                LOOP WHILE INSTR("IVX", ch$)

                                Numb$ = MID$(Txt$, k - 1, m1 - 1)
```

```
                        '— translate the Roman numeral(s) to Arabic numerals
                        NumFound = 30  '— there are 30 Roman numbers to check
                        CALL FindExact(VARPTR(Numbers$(1)), NumFound, Numb$)

                        IF NumFound <> -1 THEN

                                NewWord$ = "za" + Num$(NumFound + 1)
                                m1 = INSTR(k, Txt$, " ")

                                IF m1 = 0 THEN
                                        Txt$ = LEFT$(Txt$, m - 1) + NewWord$
                                ELSE
                                        Txt$ = LEFT$(Txt$, m - 1) + NewWord$ + RIGHT$(Txt$, LEN(Txt$) - m1 + 1
                                END IF

                        END IF

                END IF   '— are we searching for an Article =?

        END IF   '— there's a number after the Section/Article

    END IF   '— did we find a variation of Section or Article?

END IF   '— INSTR(Text,"Sec.") was found
NextStep:

        Start = m + 1

    LOOP UNTIL m = 0

    SearchStr$ = "Art"
    Letter$ = "za"

NEXT

'— start looking at the beginning of the line
Start = 1

DO
    '— look for the section symbol
    m = INSTR(Start, Txt$, CHR$(21))

    '— if we found one, process it
    IF m THEN

        '— position right after the symbol
        k = m + 1
        m1 = 0

        '— loop until it's not a number (a space is ok, however)
        '   or we've reached the end of the string
        DO
            ch = MidChar(Txt$, k + m1)
            m1 = m1 + 1
            IF k + m1 - 1 > LEN(RTRIM$(Txt$)) THEN EXIT DO
        LOOP UNTIL (ch < ASC0 OR ch > ASC9) AND ch <> 32

        '— the number is the position from right after the symbol (k)
        '   to the non-number position found in the loop above (m1 - 1)
        Numb$ = QPTrim$(MID$(Txt$, k, m1 - 1))

        IF VAL(Numb$) <= 3000 AND VAL(Numb$) > 0 THEN

            NewWord$ = "zs" + Numb$
            m1 = INSTR(k + 1, Txt$, " ")

            IF m1 THEN
                Txt$ = LEFT$(Txt$, m - 1) + NewWord$ + MID$(Txt$, m1)
            ELSE
                Txt$ = LEFT$(Txt$, m - 1) + NewWord$
            END IF

        END IF

        '— start looking at the next position
        Start = m + 1

    END IF

'— loop until we don't find any more section symbols
LOOP UNTIL m = 0

END SUB

SUB WindMgr (ULRow, ULCol, LRRow, LRCol, Frame, BoxColr, TextColr, Text$) STATIC

CALL Box0(ULRow - 1, ULCol - 1, LRRow + 1, LRCol + 1, Frame, BoxColr)
CALL ClearScr0(ULRow, ULCol, LRRow, LRCol, BoxColr)
CALL QPrintRC("[" + Text$ + "]", ULRow - 1, ULCol + 1, TextColr)

END SUB

SUB WordParse (TextHandle, Lines, WordHandle, Words&) STATIC

DIM Temp80 AS STR80' for retrieving lines of text from the file
DIM Temp32 AS STR32' for saving Words in the word list

LENTemp80 = LEN(Temp80)
LENTemp32 = LEN(Temp32)

Words& = 0
CALL DispMsg("Parsing line #:      word #:     ", r, c)
```

```
FOR i = 1 TO Lines

        QPrintRC STR$(i), r, c - 18, -1
        EmsGet1El Temp80, LENTemp80, i, TextHandle
        t$ = RTRIM$(Temp80.Str)
        TotW = InCount%(t$, " ") + 1' number of words in current line
        FOR Word = 1 TO TotW
                CALL Extract(t$, " ", Word, Start, SLen)
                IF SLen > 0 THEN
                        w$ = MID$(t$, Start, SLen)
'If there is chr(255), i.e. this is the first word in the sentence extract it
SecondPart:
                        IF Lang$ = "ENGLISH" THEN
                                n = INSTR(w$, CHR$(255))
                                IF n THEN
                                        w1$ = MID$(w$, n + 1)
                                        w$ = LEFT$(w$, n)
                                        WLen = LEN(w$)
                                        'put "/" before marker (chr$(24))
                                        IF WLen < 4 THEN
                                                w$ = LEFT$(w$, WLen - 1) + STRING$(4 - WLen, "/") + CHR$(255)
                                        ELSEIF MID$(w$, WLen - 2, 2) = "'s" THEN w$ = LEFT$(w$, WLen - 3) + CHR$(255)
                                        ELSEIF MID$(w$, WLen - 1, 1) = "'" THEN w$ = LEFT$(w$, WLen - 2) + CHR$(255)
                                        END IF
                                END IF
                         END IF

                        'fill out 1 and 2 char words with /'s
                        SLen = LEN(w$)
                        IF SLen < 3 THEN w$ = w$ + STRING$(3 - SLen, "/")
                        ' allow only words that start with alphabetic chars "a"-"z"
                        ASCw = ASC(w$)
                        IF (ASCw >= ASCa AND ASCw <= ASCz) OR (ASCw >= ASCupperA AND ASCw <= ASCupperZ) THEN
                                Words& = Words& + 1
                          'QPrintRC STR$(Words&), r, c - 5, -1
                                IF Lang$ = "ENGLISH" THEN
                                        'the following doesn't apply to German ----------------
                                        IF RIGHT$(w$, 2) = "'s" THEN ' remove the 's
                                                w$ = LEFT$(w$, SLen - 2)
                                        ELSEIF RIGHT$(w$, 1) = "'" THEN
                                                w$ = LEFT$(w$, SLen - 1)
                                        END IF
                                        '-----------------------------------------------------
                                END IF

                                'store the word in EMS now
                                Temp32.Str = w$
                                EmsSet Temp32, LENTemp32, Words&, WordHandle

                                IF EmsErrorX THEN '-- probably ran out of storage in EMS
                                        Words& = Words& - 1
                                        EXIT FOR
                                END IF

                        END IF
                IF w1$ <> "" THEN w$ = w1$: w1$ = "": GOTO SecondPart
                END IF
        NEXT' word in current line

        IF EmsErrorX THEN EXIT FOR

NEXT' line of text
CALL DispMsg("", r, c)
CALL DispStat(LTRIM$(STR$(Words&)) + " words were found.")

END SUB

SUB WriteIDFText STATIC

'*************************************************************************

'load in Code-->Word dictionary directly into EMS
'LoadFile$ = LstDir$ + "DICT.WRD"
'DictWordNum = FileSize&(LoadFile$) \ SixtyFour
'CALL DispMsg("Loading " + Num$(DictWordNum) + " Dictionary Entries", r, c)
'DIM DictWordTemp AS DictType
'NumPages = DictWordNum \ 256 + 1
'CALL EmsAllocMem(NumPages, DictWordEMSX)
'IF EmsErrorX THEN PRINT "Couldn't allocate"; NumPages * SixteenK; "bytes of EMS.": STOP

'CALL FOpen(LoadFile$, DictFILEX)
'FOR i = 1 TO DictWordNum
'    CALL FGetT(DictFILEX, DictWordTemp, LEN(DictWordTemp))
'    CALL EmsSet1El(SEG DictWordTemp, LEN(DictWordTemp), i, DictWordEMSX)
'NEXT
'CALL FClose(DictFILEX)
'CALL DispMsg(Nul$, 0, 0)
'
'CALL DispMsg("Saving IDF.TXT and DOCKEYS.TXT for examination.", r, c)
''save arrays for examination later
'OPEN LstDir$ + "IDF.TXT" FOR OUTPUT AS 1
'FOR i = 1 TO UBOUND(IDF)
'    CALL EmsGet1El(SEG DictWordTemp, LEN(DictWordTemp), i, DictWordEMSX)
'    PRINT #1, DictWordTemp.Str; IDF(i)
'NEXT
'CLOSE #1
'
'OPEN LstDir$ + "DOCKEYS.TXT" FOR OUTPUT AS 1
'CALL FOpen(LstDir$ + "DOCKEYS.AH", DocKeysHX)
'FOR i8 = 1 TO CurrDocKey%
```

```
'   CALL FGetRT(DocKeysHC, DocKeyTemp, iB, LENDocKeyTemp)
'   CALL EmsGet1El(SEG DictWordTemp, LEN(DictWordTemp), DocKeyTemp.Code, DictWordEMS%)
'   PRINT #1, DictWordTemp.Str; DocKeyTemp.Freq
'NEXT
'CALL FClose(DocKeysHC)
'CLOSE #1

'ERASE IDF

'CALL EmsRelMem(DictWordEMS%)

'CALL DispMsg("", 0, 0)

'CALL Chime(10)

END SUB

SUB ZZZRevisionLog STATIC

'$Log:   D:/aim/aim.ba_  $
'--
'--      Rev 2.10   06 Feb 1992 13:30:14
'--
'--  [Changed by TMY:]
'--
'--  * Config Modified: Returns the Machine # and First/Last document
'--    Number to process (entered on the command line) so that only one
'--    AIM.EXE is needed instead of one for each machine.
'--
'--  * [MAIN] Modified: Changed when writing out any machine specific file
'--    (IDF.$IN, STATS.TX*, etc.) to use the machine number instead of a
'--    hard-coded number.
'--
'--  * FindSingKey Modified: Changed the check for a final slash (/) from
'--    using RIGHT$(CurrKey$, 1) to use MIDCHARX(CurrKey$, SLen)=ASC("/").
'--
'--  * LoadIntoEMS Added: loads a file quickly into EMS.
'--
'--  * EMSAlloc Added: used by LoadIntoEMS -- allocated EMS and displays an
'--    error if insufficient EMS space.
'--
'--  * Fixed bug where Start variable was COMMON SHARED and was used in
'--    multiple procedures which affected one another.
'--
'--  * Document Files (*DOC.TXT and *DOC.NDX) and configuration (*.CFG,
'--    [LANGUAGE].LST) are now opened ACCESS READ SHARED.
'--
'--  [Changed by VN:]
'--
'--  * ReadSection Added: looks for references to Section or Article
'--    numbers and "normalizes" their appearance by adding a ZS prefix for
'--    Sections and ZA for articles to the number (article numbers are
'--    converted from Roman numerals to Arabic) so that they can easily be
'--    recognized by the FindSingKey routine.
'--
'--  * ReadEnglishText Modified: added call to ReadSection before any
'--    parsing or stripping of characters is done.
'--
'--  * FindSingKey Modified: added check for section numbers (denoted by a
'--    ZS prefix to a number) and article numbers (ZA prefix) which returns
'--    a code of the Section number plus 10,563 (10,563 is the number of
'--    dictionary entries of words prior to the addition of the section and
'--    article numbers) or plus 13,563 for Article numbers.
'--
'--  * ReadEnglishText Modified: Change the StripRange code so that numbers
'--    would NOT be stripped out. [TMY Bug Fix:] Added in the
'--    Txt$ = LEFT$(Txt$, TLen) code which was left out (it's required
'--    since StripRange doesn't change the length of a string) and caused
'--    problems because the string could contain garbage at the end.
'--
'--
'--      Rev 2.9   24 Jan 1992 17:33:52
'--
'--  final 3-letter table version as of 7/1/91 by Ted M. Young
'--  uses the KEYWORD.TBL & KEYCOMB.TBL 3-letter indexes
'--
'--
'--      Rev 2.8   24 Jan 1991 10:43:38
'--
'--  Changed document loading to use the ISAMed text file (i.e., a single
'--  large file instead of many individual files).
'--
'--  EMSGet/Set are used instead of the paged EMSGet1El/Set1El access using
'--  the GetDocKey/SetDocKey SUBs.
'--
'--  COMBKEY.STR & SINGKEY.STR files are loaded directly into EMS since
'--  they are to large to load into normal memory.
'--
'--
'--      Rev 2.7   01 Aug 1990 15:06:16
'--
'--  Renased CheckCombKey to FindCombKey to be more consistent with naming.
'--
'--  Fixed bug with non-wildcard words (i.e., those that end with a slash
'--  "/") where it would try and compare the word in the document with the
'--  word in the keyword list, but without removing the slash from the
'--  keyword, so the comparison would never be true.
'--
'--  Added feature where the FindCombKey would test for a slash for each
'--  word in the combined keyword.
```

```
'--
'--
'--     Rev 2.6   26 Jul 1990 16:29:52
'--
'-- Fixed bug where it wouldn't read the correct information from the
'-- configuration (.CFG) file.  It wasn't reading the NdxDir$, so it would
'-- stop with a bad file name error.
'--
'--
'--     Rev 2.5   25 Jul 1990 18:20:28
'--
'-- Fixed bug: Sometimes the LineCount routine will return too many lines,
'-- so a check for EOF was put in before reading each line.
'--
'--
'--     Rev 2.4   13 Jul 1990 13:14:56
'-- Changed CONFIG so that it can read a configuration File$ named on
'-- the command line, but will still default to reading AIM.CFG.
'--
'--
'--     Rev 2.3   13 Jul 1990 11:22:56
'--
'-- BC/PDS 7.00 Compatible, use of static arrays in typed storage
'-- for DocKeys

END SUB

'=============
'  AIMPASS2.BAS
'=============

'Invokation: AIMPASS2 ConfigFile
'------------

'Creates: Key.Ndx, Weight.Ndx, \Aim\[Command$].Dat
'--------

'Uses: Dict.Wrd, Idf.Sin, DocIndex.Ah, DocKeys.Ah
'-----
' nth Rec of Key.Ndx contains Number of Keywords in Document
'     followed by 127 codes

' TYPE KeyNdx127
'    Num AS INTEGER
'    Code(1 TO 127) AS INTEGER

' nth Rec of Weight.Ndx contains 127 Salton Weights
'     computed with the formula below .

'  TYPE WeightNdx
'    Weight(1 TO 127) AS SINGLE

'SALTON WEIGHT FORMULA
'--------------------

'                Log2(FreqinDoc+1)*Log2((TotDocs*1.5)/DocsWithWord+3+TotDocs*0.001)
'Weight(Word)= ------------------------------------------------------------------
'                                Log2 (2 + TotalKWordsInDoc / 10)

DEFINT A-Z

'$INCLUDE: '\\VADIM\C-DRIVE\USER\INCLUDE\TYPES.BI'

TYPE WeightCode
        Code AS INTEGER
        Wt AS SINGLE
END TYPE

TYPE Flen
        Str AS STRING * 12
END TYPE

TYPE DocIndexType
        Ndx AS LONG
        Num AS INTEGER
        Tot AS LONG
        Padding AS STRING * 6
END TYPE

TYPE SmallDocIndexType
        Ndx AS LONG
        Num AS INTEGER
        Tot AS LONG
END TYPE

TYPE STR49
        Str AS STRING * 49
END TYPE

TYPE DocKeyType
        Code AS INTEGER
        Freq AS INTEGER
END TYPE

DECLARE SUB Config (Machine$, First, Last)
DECLARE SUB DispMsg (Msg$, r%, c%)
DECLARE SUB WindMgr (ULRow%, ULCol%, LRRow%, LRCol%, Frame%, BoxColr%, TextColr%, Text$)
DECLARE FUNCTION LoadIntoEMS% (File$)
DECLARE FUNCTION Log2! (x!)
DECLARE FUNCTION Num$ (x)
DECLARE FUNCTION Ems$ (Handle%, Element%, Size%)
```

```
'------------------------------------------------------ External routines

'$INCLUDE: '\\VADIM\C-DRIVE\USER\INCLUDE\DECLARES.BI'

'------------------------------------------------------ PROGRAM START

CONST Sing = 0, Comb = 1
COMMON SHARED ReProcess, Fg, Bg, NormAttr, RevAttr
COMMON SHARED DocDir$, KeyDir$, LstDir$, AtList$, Lang$, NdxDir$
COMMON SHARED XLateTable%(), SingTableX(), CombTableX()
COMMON SHARED ThirtyTwo, SixtyFour, SixteenK, ThirtyTwoK&
COMMON SHARED Log2Const!, Threshold!
COMMON SHARED Ascend, Descend, FALSE, TRUE
COMMON SHARED Enter, Escape

'$$$keyword-flag$$$ "Last Author: %v  Revision: %v  Date: %f"
CONST Version$ = "Last Author: TED  Revision: 18  Date: 8-Sep-92,18:31:06"
Log2Const! = LOG(2!)
SixteenK = 16384
ThirtyTwoK& = 32768
SixtyFour = 64
ThirtyTwo = 32
Ascend = 0
Descend = 1
FALSE = 0
TRUE = NOT FALSE
Enter = 13
Escape = 27

ThresholdWord = 80

DIM AvgWeight AS SINGLE '-- average of the ThresholdWord keywords' weights

'------------------------------------------------------------------------

Config Machine$, FirstDoc, LastDoc

QPrintRC "AIMPASS2 Started At " + TIME$, 1, 25, -1
Col = (80 - LEN(Version$)) \ 2
QPrintRC Version$, 2, Col, -1
LOCATE 4, 1

'-- load in Code-->Word dictionary directly into EMS
LoadFile$ = LstDir$ + "DICT.WRD"
DIM DictWrdTemp AS DictType
DictWrdNum = FileSize&(LoadFile$) \ LEN(DictWrdTemp)
DictWrdEMS = LoadIntoEMS(LoadFile$)

'-- load the IDF.SIN file
LoadFile$ = LstDir$ + "IDF.SIN"
IDFNum = FileSize&(LoadFile$) \ 4
PRINT "Loading "; LoadFile$; IDFNum
REDIM IDF!(1 TO IDFNum)
REDIM IDFTemp(1 TO IDFNum) AS LONG
FGetAH LoadFile$, SEG IDFTemp(1), LEN(IDFTemp(1)), IDFNum

'-- Store Average Doc Frequency in {COMMAND$}.DAT file in LST directory
FOR i = 1 TO IDFNum
        IF IDFTemp(i) > 0 THEN
                SumOfAllFreq& = SumOfAllFreq& + IDFTemp(i)
                NumWordsUsed = NumWordsUsed + 1
        END IF
NEXT

AvgDocFreq! = SumOfAllFreq& / NumWordsUsed

OPEN NdxDir$ + "AVGDOCFC.DAT" FOR OUTPUT AS #7
        PRINT #7, STR(AvgDocFreq!)
CLOSE #7

LoadFile$ = LstDir$ + "DCCINDEX.AH"
DIM DocNdxTemp AS DocIndexType
Count& = FileSize&(LoadFile$) \ LEN(DocNdxTemp)
PRINT "Loading "; LoadFile$; Count&
DIM DocNdx AS DocIndexType
LenDocNdx = LEN(DocNdx)
FOpenAll LoadFile$, 0, 4, LoadHand

LoadFile$ = LstDir$ + "DOCKEYS.AH"
NumDocKeys& = FileSize&(LoadFile$) \ 4
PRINT "Opening "; LoadFile$; NumDocKeys&
CALL FOpenAll(LoadFile$, 0, 4, DocKeys%)' open up the file for use later with FGetRT's
DIM DocKeyTemp AS DocKeyType
DocKeyLEN = LEN(DocKeyTemp)

'========================================================================

CALL DispMsg("2nd Pass: Calculating Inverse Document Frequencies", r, c)

CountLog! = Log2!(CSNG(Count&))

Log2of5! = Log2!(5!)
A& = FRE("")
WordUse$ = STRING$(DictWrdNum \ 8 + 1, 0)' setup and clear bit array -- indicates which words were used

MaxDF! = CLNG(Count&) * 1.5   '"TotDocs * 1.5"

IDFADD! = 3 + .001 * Count&   '"DocFreq + 3 + (TotDocs * 0.001)"
FOR Word = 1 TO DictWrdNum
        IF IDFTemp(Word) > 0 THEN
                IDF!(Word) = Log2!(MaxDF! / (IDFTemp(Word) + IDFADD!))
        END IF
```

```
NEXT

ERASE IDFTemp

CALL DispMsg("", 0, 0)

CALL DispMsg("2nd Pass: Calculating Formula Weights:      ", r, c)

OPEN LstDir$ + "Weights.Lst" FOR OUTPUT AS 1

DIM KeyNdxRec AS KeyNdx127
DIM WeightNdxRec AS WeightNdx127

OPEN NdxDir$ + "KEY.NDX" FOR RANDOM AS #3 LEN = LEN(KeyNdxRec)
OPEN NdxDir$ + "WEIGHT.NDX" FOR RANDOM AS #4 LEN = LEN(WeightNdxRec)

TotDocLen& = 0

AvgWeight = 0

Start& = 1

FOR DocNum& = Start& TO Count&

        FGetRT LoadHand, DocNdx, DocNum&, LenDocNdx
        A$ = STR$(DocNum&)

        TotDocLen& = TotDocLen& + DocNdx.Tot

        '-- output every document to the WEIGHTS.LST File
        'PRINT #1, A$; " ---------------------------------- Tot Freq:"; DocNdx.Tot
        'PRINT #1, USING "\  \ \   \ \    \ Code_# &"; "Freq"; "DocFr"; "Weight"; "Keyword"

        KeyNdxRec.Num = 0
        NdxNum = DocNdx.Num
        IF NdxNum = 0 THEN
                PRINT #1, "No Keywords for document #"; A$
                GOTO Pass2Skip
        END IF
        CALL QPrintRC(STR$(DocNum&), r, c - 6, -1)

        REDIM Weight(1 TO NdxNum) AS WeightCode

        ' total number of term appearances in this document, i.e., if 2 terms
        ' appear 10 times each, then this number would be 20, as opposed to
        ' .Num which would be 2 in this case.
        Tot! = DocNdx.Tot
        IF Tot! < 0 THEN
                Chime 2
                Tot! = ABS(Tot!)
        END IF

        IF Tot! = 0 THEN ' shouldn't happen!
                CALL Chime(10)
                PRINT #1, "There were no Term appearances in document "; A$
                PRINT #1, "However, there were"; NdxNum; "keywords."
                i$ = INPUT$(1)
                GOTO Done
        END IF

        TotLog2! = Log2!(2! + Tot! / 10!)   '"Log[(TotWords in Doc/10)+2]

        FOR Word = 1 TO NdxNum
                Index& = Word + DocNdx.Ndx - 1
                FGetRT DocKeysHX, DocKeyTemp, Index&, DocKeyLEN
                Freq! = DocKeyTemp.Freq' frequency of word in current document

                '-- account for integers > 32767 which went negative
                IF Freq! < 0 THEN
                        Freq! = Freq! + 65536
                END IF
                Freq! = Freq! + 1
                Code = DocKeyTemp.Code
                Weight(Word).Code = Code
                Weight(Word).Wt = Log2!(Freq!) * IDF!(Code) / TotLog2!

                'PRINT #1, USING "#### ##### #.#### ##### &"; Freq! - 1; IDFTemp(Code); Weight(Word).Wt; Code; Ems$(DictWrdEMS%, Code,
        NEXT

        'sort in descending order according to weight
        'so the keys are output in decreasing importance
        SortT Weight(1), NdxNum, Descend, LEN(Weight(1)), LEN(Weight(1).Code), -3

        '-- only add if there are ThresholdWord or more words
        '   otherwise we'd be adding zero
        IF NdxNum >= ThresholdWord THEN
                AvgWeight = AvgWeight + Weight(ThresholdWord).Wt
        END IF

        FOR k = 1 TO 127
                WeightNdxRec.Weight(k) = 0
        NEXT

        FOR Word = 1 TO NdxNum
                IF Weight(Word).Wt > Threshold! THEN
                        IF KeyNdxRec.Num < 127 THEN
                                Code = Weight(Word).Code
                                CALL SetBit(WordUse$, Code, 1)
                                KeyNdxRec.Num = KeyNdxRec.Num + 1
                                KeyNdxRec.Code(KeyNdxRec.Num) = Code
```

```
                        WeightNdxRec.Weight(KeyNdxRec.Num) = Weight(Word).Wt
                        IF WeightNdxRec.Weight(KeyNdxRec.Num) < Threshold! THEN STOP
                ELSE
                        NumOver127% = NumOver127% + 1
                        KeyNdxRec.Num = KeyNdxRec.Num + 1
                        WeightOver127! = WeightOver127! + Weight(Word).Wt
                END IF
                'PRINT #1, USING "###: #.#### &"; KeyNdxRec.Num; Weight(Word).Wt; Ems$(DictWrdEMS%, Weight(Word).Code, LEN(Dic
        END IF
    NEXT

Pass2Skip:

    PUT #3, DocNum&, KeyNdxRec
    PUT #4, DocNum&, WeightNdxRec

    Totk& = Totk& + KeyNdxRec.Num
    IF INKEY$ = CHR$(27) THEN EXIT FOR

NEXT

'-- Calculate Threshold based on avg of the ThresholdWord keyword's Salton Weight
Threshold! = AvgWeight / Count&

'-- now go through the key.ndx and change the key count (KeyNdx.Num) so
'   that only the words that have weight >= the threshold will be used

FOR DocNum& = Start& TO Count&

    GET #3, DocNum&, KeyNdxRec
    GET #4, DocNum&, WeightNdxRec

    '-- look through the weights until we find one that's lower than
    '   the threshold
    NewNdxNum = 0
    FOR i = 1 TO KeyNdxRec.Num
        IF WeightNdxRec.Weight(i) < Threshold! THEN
            '-- this one is lower than the threshold, so the previous
            '   word should be the last keyword for this document
            NewNdxNum = i - 1
            EXIT FOR
        END IF
    NEXT

    IF NewNdxNum > 0 THEN '-- save new Ndx Num keyword count
        KeyNdxRec.Num = NewNdxNum
        PUT #3, DocNum&, KeyNdxRec
    END IF

NEXT

CALL DispMsg("", 0, 0)

CLOSE 3, 4' close the random-access keycoded file
CALL FClose(DocKeys%)
CALL EmsRelMem(DictWrdEMS%)

PRINT #1, "Total keys written:"; Totk&
PRINT #1, "Average # keys/record:"; Totk& / Count&
PRINT #1, "Average Document Length:"; TotDocLen& / Count&
PRINT #1, ""
PRINT #1, "Threshold:"; Threshold!
PRINT #1, ""

WordsInUse% = 0
FOR i = 1 TO DictWrdNum
    WordsInUse% = WordsInUse% - GetBit%(WordUse$, i)
NEXT

PRINT #1, "Words used in collection:"; WordsInUse%; NumWordsUsed
PRINT #1, "Average Document Frequency of words used:"; AvgDocFreq!

PRINT #1, ""
PRINT #1, "Number of documents having more than 127 keywords:"; NumOver127&

Done:

CLOSE 1' close the Weights.Lst file

Chime 10

SUB Config (Machine$, First, Last) STATIC

Cmd$ = QPTrim$(COMMAND$)

IF COMMAND$ <> "" THEN
    ConfigFile$ = COMMAND$ + ".CFG"
ELSE
    Chime 10
    PRINT "No database specified."
    END
END IF

'IF NOT Exist(ConfigFile$) THEN
'    Chime 10
'    PRINT "File"; ConfigFile$; " was not found "
'    END
'END IF
OPEN ConfigFile$ FOR INPUT AS #1
INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$, Threshold!
```

```
CLOSE #1

COLOR Fg, Bg, Brdr: CLS

NormAttr = OneColorX(Fg, Bg)
RevAttr = OneColorX(Bg, Fg AND 7)

IF NOT EmsLoadedX THEN
        PRINT "No EMS!"
        BEEP
        END
END IF

END SUB

SUB DispMsg (Msg$, r, c) STATIC
STATIC WindOpen, ScrX() ' is there already a message displayed?
SHARED Fg, Bg

IF Msg$ = "" AND WindOpen THEN GOSUB MsgClose: EXIT SUB
IF WindOpen THEN
        CALL Chime(9)
        OPEN "DEBUG" FOR OUTPUT AS 9
        PRINT #9, "WindOpen="; WindOpen; HEX$(WindOpen)
        PRINT #9, "Msg$= |"; Msg$; "|"
        PRINT #9, "TRUE="; TRUE; HEX$(TRUE)
        PRINT #9, "FALSE="; FALSE; HEX$(FALSE)
        CLOSE 9
        CALL Chime(8)
        CLS
        END
        i$ = INPUT$(1)
        GOSUB MsgClose
END IF

Wid = LEN(Msg$)
IF Wid > 50 THEN Wid = 50

Msg$ = Msg$ + " " ' make sure there's a space to find at the end (see below)

MaxLin = LEN(Msg$) \ Wid + 3
IF MaxLin > 23 THEN MaxLin = 23
REDIM Text$(MaxLin)
Lin = 0
DO
        Lin = Lin + 1' incr current lin # (also element in text display array)
        LastSpc = QInstrBC(Wid + 1, Msg$, " ")' look for the last space so we can word wrap
        Text$(Lin) = LEFT$(Msg$, LastSpc - 1)
        Msg$ = MID$(Msg$, LastSpc + 1)'remove portion of string that's in t$
LOOP WHILE LEN(Msg$) > Wid

Msg$ = RTRIM$(Msg$)
IF LEN(Msg$) THEN
        Lin = Lin + 1
        Text$(Lin) = Msg$
END IF

IF r <> 0 AND c = 0 THEN
        ULr = r
ELSE
        ULr = 9
END IF
        DULr = ULr - 1
        LRr = ULr + Lin + 1
        DLRr = LRr + 2

horizmargin = (80 - Wid) \ 2
        ULc = horizmargin
        DULc = ULc - 3
        LRc = 80 - INT(horizmargin)
        IF Wid / 2 = Wid \ 2 THEN LRc = LRc + 1
        DLRc = LRc + 1

REDIM ScrX(ArraySizeX(DULr, DULc, DLRr, DLRc))
CALL ScrnSaveO(DULr, DULc, DLRr, DLRc, SEG ScrX(0))
CALL WindMgr(ULr, ULc, LRr, LRc, 4, NormAttr, RevAttr, "Status")
FOR i = 1 TO Lin
        CALL QPrintRC(Text$(i), ULr + i, ULc + 1, -1)
NEXT

r = ULr + Lin
c = ULc + 1 + LEN(Text$(Lin))
IF LEN(Text$(Lin)) + 2 = Wid THEN c = ULc + 1: r = r + 1

ERASE Text$
WindOpen = TRUE

EXIT SUB

'------------------------------------------------------------ close window
MsgClose:

CALL ScrnRestO(DULr, DULc, DLRr, DLRc, SEG ScrX(0))
ERASE ScrX
WindOpen = FALSE
RETURN

END SUB

FUNCTION Ems$ (Handle, Element, Size) STATIC
```

```
DIM Temp32 AS STR32
DIM Temp49 AS STR49

IF SizeX = 32 THEN
        CALL EmsGet1El(SEG Temp32, Size, Element, Handle)
        Ems$ = RTRIM$(LEFT$(Temp32.Str, 30))
ELSEIF SizeX = 49 THEN
        CALL EmsGet1El(SEG Temp49, Size, Element, Handle)
        Ems$ = RTRIM$(LEFT$(Temp49.Str, 47))
ELSE
        STOP
END IF

END FUNCTION

FUNCTION LoadIntoEMS (File$) STATIC

'———— Returns the handle where the file was loaded into ————

EMSPg = EmsGetPFSeg%
SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAllocMem NumPages, FileEMS

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
FOpenAll File$, 0, 4, LoadFile
FOR i = 1 TO Num32kBlocks + 1

        Box0 14, 10, 18, 70, 2, RevAttr
        PaintBox0 14, 10, 18, 70, RevAttr
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1) + " ", 16, 12, RevAttr

        '— map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT

        '— seek to beginning of current block
        FSeek LoadFile, (i - 1) * ThirtyTwoK&

        IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        IF i < Num32kBlocks + 1 THEN

                '— get the 32k block and put it directly into the EMS page frame
                FGetA LoadFile, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        ELSE

                '— load the left over (<32k) bytes
                FGetA LoadFile, BYVAL EMSPg, BYVAL 0, LeftOver&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        END IF
NEXT

FClose LoadFile

ClearScr0 14, 10, 18, 70, NormAttr

LoadIntoEMS = FileEMS

END FUNCTION

FUNCTION Log2! (x!) STATIC

SHARED Log2Const!

Log2! = LOG(x!) / Log2Const!

END FUNCTION

FUNCTION Num$ (x) STATIC

Num$ = LTRIM$(STR$(x))

END FUNCTION

SUB WindMgr (ULRow, ULCol, LRRow, LRCol, Frame, BoxColr, TextColr, Text$) STATIC

CALL Box0(ULRow - 1, ULCol - 1, LRRow + 1, LRCol + 1, Frame, BoxColr)
CALL ClearScr0(ULRow, ULCol, LRRow, LRCol, BoxColr)
CALL QPrintRC("[" + Text$ + "]", ULRow - 1, ULCol + 1, TextColr)

END SUB

DEFINT A-Z
```

```
'==========
' KYINVERT.BAS
'==========

'Invoked: KYINVERT ConfigFile
'--------

'Creates: KyInvert.Ndx, KyInvert.Dat
'--------

'Uses: Key.Ndx, Weight.Ndx & Dict.Wrd for NumKeys
'----

' nth Rec of KyInvert.Ndx contains nth Code,
' ptr into KyInvert.Dat & Number of Docs indexed with this word
'  TYPE NdxType
'     Code AS INTEGER
'     Index AS LONG
'     Num AS INTEGER

' Records are pointed to by .Index of KyInvert.Dat
' .Rec contains Document that is indexed with this word
' .Value 1000 * Salton Weight In Document
'  TYPE KeyInfoLONG
'     Rec AS LONG
'     Value AS INTEGER

'$INCLUDE: '\\vadim\c-drive\user\include\types.bi'

'-- now in types.bi above
'TYPE KeyInfoLONG
'   Rec AS LONG       '-- record (document) number
'   Value AS INTEGER  '-- holds the weight of each keyword in the record
'                     '   scaled to fit an integer
'END TYPE

TYPE LinkHead
        Num AS LONG       '-- number of nodes in the linked list
        FirstPtr AS LONG  '-- pointer to first node
        LastPtr AS LONG   '-- pointer to the last node
        Pad AS STRING * 4 '-- padding for huge arrays
END TYPE

'-- holds the actual data, in this case the Keyword's record # and its value
TYPE LinkNode
        Info AS KeyInfoLONG
        Ptr AS LONG
END TYPE

'-- Information for VMS routines
TYPE VMSTableType
        Handle AS INTEGER
        TempFile AS STRING * 62
END TYPE

TYPE FileInfoType
        Year   AS INTEGER
        Month  AS INTEGER
        Day    AS INTEGER
        Hour   AS INTEGER
        Minute AS INTEGER
        Second AS INTEGER
        Size   AS LONG
        Attrib AS INTEGER
END TYPE

CONST NULL = 0

' Color Attributes
COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr, ShiftValueX, MaxWgt

' Directories
COMMON SHARED LstDir$, DocDir$, KeyDir$, NdxDir$, ConfigName$

' Temp variables
COMMON SHARED NodeLEN%, NodeTemp AS LinkNode


'-- global EMS usage flag
COMMON SHARED gfEMS AS INTEGER

'-- number of allocations for VMS routines
COMMON SHARED gVMSNumAllocations AS INTEGER
COMMON SHARED VMSError%

'-- allocation information (handle and filename)
COMMON SHARED gVMSTable() AS VMSTableType


'$INCLUDE: '\\vadim\c-drive\user\include\const.bi'

'------------------------------------------------------------ Internal SUBs

DECLARE SUB AddNode (Head AS LinkHead, Info AS KeyInfoLONG, hStorage AS INTEGER, FreePtr AS LONG)
DECLARE SUB Config ()

'------------------------------------------------------------ Internal FUNCTIONs

DECLARE FUNCTION EmsAllocX (NumBytes$, HandleX)
DECLARE FUNCTION FileDate$ (FInfo AS ANY)
```

```
DECLARE FUNCTION Unique$ (Path$)
DECLARE FUNCTION VMSAlloc% (NumBytes&, HandleX)
DECLARE FUNCTION VMSReleaseX% (HandleX)

'------------------------------------------------------- External SUBs

DECLARE SUB Chime (NumberX)
DECLARE SUB DeleteT (SEG StartElement AS ANY, ElSizeX, NumElsX)
DECLARE SUB EmsAllocMem (NumPagesX, HandleX)
DECLARE SUB EmsGet (SEG Value AS ANY, ElSizeX, ELNum&, HandleX)
DECLARE SUB EmsSet (SEG Value AS ANY, ElSizeX, ELNum&, HandleX)
DECLARE SUB EmsRelMem (HandleX)
DECLARE SUB FClose (HandleX)
DECLARE SUB FCreate (FileName$)
DECLARE SUB FGetRT (HandleX, Destination AS ANY, RecNumber&, RecSizeX)
DECLARE SUB FileInfo (FileName$, SEG Address AS ANY)
DECLARE SUB FOpen (FileName$, HandleX)
DECLARE SUB FOpenAll (FileName$, AccessModeX, ShareModeX, HandleX)
DECLARE SUB FPut (HandleX, Work$)
DECLARE SUB FPutAH (FileName$, SEG Element AS ANY, ElSizeX, NumElsX)
DECLARE SUB FPutRT (HandleX, Destination AS ANY, RecNumber&, RecSizeX)
DECLARE SUB FPutT (HandleX, Source AS ANY, RecSizeX)
DECLARE SUB FSeek (HandleX, Location&)
DECLARE SUB QPrintRC (S:$, RowX, ColX, ColrX)

'------------------------------------------------------ External FUNCTIONs

DECLARE FUNCTION DOSErrorX
DECLARE FUNCTION WhichErrorX
DECLARE FUNCTION ErrorMsg$ (ErrNumX)
DECLARE FUNCTION EmsErrorX
DECLARE FUNCTION EmsLoadedX
DECLARE FUNCTION EmsPagesFreeX
DECLARE FUNCTION ExistX (FileName$)
DECLARE FUNCTION FileSize& (FileName$)
DECLARE FUNCTION OneColorX (ForeX, BackX)
DECLARE FUNCTION PeekZX (Segment, Address)

'------------------------------------------------------- PROGRAM START

'$$Keyword-flag$$$ "Last Author: Xv  Revision: Xv  Date: Xf"
CONST Version$ = "Last Author: TED  Revision: 16  Date: 9-Sep-92,18:51:04"

CALL Config

LOCATE 18, 1

QPrintRC "Key Invert (KYINVERT) Started on " + DATE$ + " at " + TIME$, 1, 15, -1
Col = (80 - LEN(Version$)) \ 2
QPrintRC Version$, 2, Col, -1

'-- use dictionary to determine total number of keywords
LoadFile$ = LstDir$ + "DICT.WRD"
DIM DictEntry AS DictType
NumKeys = FileSize&(LoadFile$) \ LEN(DictEntry)

NodeLEN = LEN(NodeTemp)

KyLEN = LEN(Ky)

DIM Wgt AS WeightNdx127
WgtLEN = LEN(Wgt)

DIM KeyNdxFInfo AS FileInfoType
DIM CountFInfo AS FileInfoType

'-- use KEY.NDX to determine total number of documents
LoadFile$ = NdxDir$ + "KEY.NDX"
FileInfo LoadFile$, KeyNdxFInfo
NumDocs& = FileSize&(LoadFile$) \ KyLEN
FOpenAll LoadFile$, ACCESSREAD, SHAREDENYWRITE, KeyNdxHX
IF DOSErrorX THEN
        LOCATE 18, 1
        PRINT "Opening "; LoadFile$
        PRINT "A DOS Error Occurred:"; WhichErrorX; ErrorMsg$(WhichErrorX)
        STOP
END IF

LoadFile$ = NdxDir$ + "WEIGHT.NDX"
FOpenAll LoadFile$, ACCESSREAD, SHAREDENYWRITE, WeightNdxHX
IF DOSErrorX THEN
        LOCATE 18, 1
        PRINT "Opening "; LoadFile$
        PRINT "A DOS Error Occurred:"; WhichErrorX; ErrorMsg$(WhichErrorX)
        STOP
END IF

'-- array for the KYINVERT.NDX information
REDIM Ndx(1 TO NumKeys) AS NdxType

DIM Info AS KeyInfoLONG
InfoLEN = LEN(Info)

-- count how many keywords there actually are
QPrintRC "Counting Number of Keywords in use...", 4, 1, -1
TotKeys& = 0
LoadFile$ = LstDir$ + ConfigName$ + ".CNT"
'-- make sure that the count file is at least as recent as the KEY.NDX file
IF Exist(LoadFile$) THEN
        FileInfo LoadFile$, CountFInfo
        IF FileDate$(CountFInfo) < FileDate$(KeyNdxFInfo) THEN
```

```
                '-- delete the count file because it's outdated
                KILL LoadFile$
        END IF
END IF

IF NOT Exist(LoadFile$) THEN
        FOR DocNum& = 1 TO NumDocs&
                QPrintRC STR$(DocNum&), 4, 38, -1
                FGetRT KeyNdxHX, Ky, DocNum&, KyLEN
                TotKeys& = TotKeys& + Ky.Num
                QPrintRC STR$(TotKeys&), 4, 45, -1
        NEXT
        hFile = FREEFILE
        OPEN LoadFile$ FOR OUTPUT AS #hFile
                PRINT #hFile, TotKeys&
        CLOSE #hFile
ELSE
        hFile = FREEFILE
        OPEN LoadFile$ FOR INPUT AS #hFile
                INPUT #hFile, TotKeys&
        CLOSE #hFile
END IF

'-- divide into sections and allocate maximum EMS
NumBytes& = 16384& * (EmsPagesFreeX - 1)
MaxNodes& = NumBytes& \ NodeLEN
NumSections = 4
SectionSize! = NumKeys / NumSections

'-- try to allocate space in EMS
NumPages = EmsAlloc(NumBytes&, InvertDatMEM)

IF NumPages > 0 THEN

        '-- enough EMS was available
        QPrintRC "Number of EMS Pages Allocated =" + STR$(NumPages) + ", MaxNodes =" + STR$(MaxNodes&), 6, 1, -1
        gfEMS = TRUE

ELSE

        '-- not enough EMS was available, use VMS routines
        QPrintRC "Insufficient EMS Available. Requested" + STR$(ABS(NumPages)) + " EMS pages.", 6, 1, -1
        QPrintRC STR$(EmsPagesFreeX) + " EMS pages were available.", 7, 1, -1
        gfEMS = FALSE

        '-- try to allocate space using VMS
        IF VMSAlloc(NumBytes&, InvertDatMEM) THEN

                QPrintRC "Number of bytes of VMS Allocated =" + STR$(NumBytes&), 8, 1, -1

        ELSE

                QPrintRC "Unable to allocate VMS.", 8, 1, -1
                STOP

        END IF

END IF

SaveFile$ = NdxDir$ + "KYINVERT.DAT"
FCreate SaveFile$
FOpen SaveFile$, InvertDatFile

Index& = 1

FOR Section = 1 TO NumSections

        KeyLBound = (Section - 1) * SectionSize! + 1
        KeyUBound = Section * SectionSize!
        '-- make sure process to the last key in case of round-off errors
        IF Section = NumSections AND KeyUBound <> NumKeys THEN KeyUBound = NumKeys

        REDIM LinkList(KeyLBound TO KeyUBound) AS LinkHead

        QPrintRC "Section" + STR$(Section) + ": Processing Keys" + STR$(KeyLBound) + " to" + STR$(KeyUBound), 10, 1, -1

        '-- initialize element number of the next free pointer
        FreePtr& = 1

        FOR DocNum& = 1 TO NumDocs&

                QPrintRC "Doc #" + STR$(DocNum&) + STR$(FreePtr&) + "            ", 12, 1, -1

                FGetRT KeyNdxHX, Ky, DocNum&, KyLEN
                FGetRT WeightNdxHX, Wgt, DocNum&, WgtLEN

                Info.Rec = DocNum&

                '-- process the list of keywords for this document
                FOR j = 1 TO Ky.Num

                        CurrCode = Ky.Code(j)

                        '-- only add this code to the linked list if it's
                        '   in the section we're currently processing
                        IF CurrCode >= KeyLBound AND CurrCode <= KeyUBound THEN

                                '-- if the weight is larger than the maximum (scaled) weight
                                '   then just assign it the maximum weight
                                IF Wgt.Weight(j) > MaxWgt THEN
                                        Wgt.Weight(j) = MaxWgt
                                END IF
```

```
                '-- scale the single precision weight to an integer
                Info.Value = Wgt.Weight(j) * ShiftValue%

                '-- AddNode stores the current Info at FreePtr& and increments it
                AddNode LinkList(CurrCode), Info, InvertDatMEM, FreePtr&

                '-- if FreePtr& was just incremented above the allocation then stop
                IF FreePtr& > MaxNodes& THEN
                        Chime 8
                        LOCATE 18, 1
                        PRINT "Exceeded allocation of"; MaxNodes&; "nodes."
                        PRINT "DocNum& ="; DocNum&; " j (keyword number) ="; j
                        STOP
                END IF

                END IF

        NEXT

    NEXT

    ' ---------------------------------------------------
    ' start writing inverted file by traversing the linked list
    ' ---------------------------------------------------

    '-- write out keys for the current section
    FOR i = KeyLBound TO KeyUBound

        '-- point to head of linked list for this keyword
        Ptr& = LinkList(i).FirstPtr

        QPrintRC STR$(i) + " out of" + STR$(KeyUBound) + ":" + STR$(LinkList(i).Num) + "      ", 14, 1, -1

        Ndx(i).Code = i
        Ndx(i).Index = Index&
        Ndx(i).Num = LinkList(i).Num

        '-- traverse linked-list for this keyword
        DO WHILE Ptr&

            IF gfEMS THEN
                EmsGet NodeTemp, NodeLEN, Ptr&, InvertDatMEM
            ELSE
                FGetRT InvertDatMEM, NodeTemp, Ptr&, NodeLEN
            END IF

            '-- write (append) the Info to disk
            FPutT InvertDatFile, NodeTemp.Info, InfoLEN

            '-- get next pointer
            Ptr& = NodeTemp.Ptr

        LOOP

        Index& = Index& + Ndx(i).Num

    NEXT

NEXT Section

'-- release memory for inverted list data
IF gfEMS THEN
    EmsReLMem InvertDatMEM
ELSE
    IF NOT VMSRelease(InvertDatMEM) THEN
        Chime 8
        LOCATE 22, 1
        PRINT "Problem in VMSRelease."
        STOP
    END IF
END IF

'-- close input files KEY.NDX and WEIGHT.NDX
FClose KeyNdxH%
FClose WeightNdxH%

'-- close KYINVERT.DAT output file
FClose InvertDatFile

'-- write the KYINVERT.NDX to disk from the Ndx() array
FPutAH NdxDir$ + "KYINVERT.NDX", Ndx(1), LEN(Ndx(1)), NumKeys


DATA "   Copyright 1990-2 by Ted M. Young.  ALL RIGHTS RESERVED.   "

SUB AddNode (Head AS LinkHead, Info AS KeyInfoLONG, hStorage AS INTEGER, FreePtr AS LONG) STATIC

IF Head.FirstPtr = NULL THEN

    Head.FirstPtr = FreePtr
    Head.LastPtr = FreePtr

ELSE

    IF gfEMS THEN
        EmsGet NodeTemp, NodeLEN, Head.LastPtr, hStorage
        IF EmsError% THEN
            Chime 8
            PRINT "EMS Get Error in AddNode at Head.LastPtr="; Head.LastPtr
            STOP
        END IF
```

```
        ELSE
                FGetRT hStorage, NodeTemp, Head.LastPtr, NodeLEN
        END IF

        NodeTemp.Ptr = FreePtr      '-- point the node's next ptr to new node

        IF gfEMS THEN
                EmsSet NodeTemp, NodeLEN, Head.LastPtr, hStorage
                IF EmsError% THEN
                        Chime 8
                        PRINT "EMS Set Error in AddNode at Head.LastPtr="; Head.LastPtr
                        STOP
                END IF
        ELSE
                FPutRT hStorage, NodeTemp, Head.LastPtr, NodeLEN
        END IF

        Head.LastPtr = FreePtr      '-- set the last pointer to the new node

END IF

'-- increment count of nodes for this linked list
Head.Num = Head.Num + 1

'-- store the info in the node and set its pointer to NULL
NodeTemp.Info = Info
NodeTemp.Ptr = NULL

IF gfEMS THEN
        EmsSet NodeTemp, NodeLEN, FreePtr, hStorage
        IF EmsError% THEN
                Chime 8
                PRINT "EMS Error in AddNode at FreePtr="; FreePtr
                STOP
        END IF
ELSE
        FPutRT hStorage, NodeTemp, FreePtr, NodeLEN
END IF

'-- increment pointer to next free node
FreePtr = FreePtr + 1

END SUB

SUB Config STATIC

IF COMMAND$ <> "" THEN
        ConfigFile$ = COMMAND$ + ".CFG"
        ConfigName$ = COMMAND$
ELSE
        PRINT "No configuration file was given."
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF
IF ExistX(ConfigFile$) THEN
        OPEN ConfigFile$ FOR INPUT AS #1
                INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$, x$
        CLOSE #1
ELSE
        CALL Chime(6)
        PRINT "Configuration file "; ConfigFile$; " Does not exist."
        DO: LOOP UNTIL LEN(INKEY$)
        END
        END
END IF

COLOR Fg, Bg, Brdr

NormAttr = OneColorX(Fg, Bg)
RevAttr = OneColorX(Bg, Fg AND 7)
```

```
ShiftValue% = 1000
MaxWgt = 32767 \ ShiftValue%

CLS

IF NOT EmsLoaded% THEN
        PRINT "No EMS!"
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF

END SUB

FUNCTION EmsAllocX (NumBytes&, HandleX) STATIC

'-- calcuate number of 16K EMS pages to allocate
NumPgs% = NumBytes& \ 16384 + 1

'-- return number of pages allocated, 0 if insufficient EMS available
IF EmsPagesFreeX < NumPgs% THEN
        EmsAllocX = -NumPgs%
ELSE
        EmsAllocMem NumPgs%, HandleX
        EmsAllocX = NumPgs%
END IF

END FUNCTION
```

```
        TempFile$ = RTRIM$(gVMSTable(Found).TempFile)
        KILL TempFile$

        DeleteT gVMSTable(Found), LEN(gVMSTable(Found)), gVMSNumAllocations - Found

        gVMSNumAllocations = gVMSNumAllocations - 1
        REDIM gVMSTable(1 TO gVMSNumAllocations) AS VMSTableType

        VMSRelease = TRUE

ELSE  '-- something's wrong!

        VMSError% = 131'-- invalid handle
        VMSRelease = FALSE

END IF

END FUNCTION

DEFINT A-Z

'FreqComp version 4.5 3/25/91 15:30
'-------------------------------

' Implements the Inverted Index access method along with the weighted values
' to calculate the frequent companions for each of the words used in the
' document collection.

'Invoked: freqcomp ConfigFile
'Creates: FreqComp.127
'Uses: KyInvert.Ndx, KyInvert.Dat, Key.Ndx, Weight.Ndx
'nth Rec of FreqComp.127 contains nth Code, Num of companions of this code &
'
'127 pairs of (code,100*weight)
'  TYPE FreqComp127
'      Num AS INTEGER
'      Code AS INTEGER
'      Comp(1 TO 127) AS INTEGER
'      Value(1 TO 127) AS STRING * 1

CONST ACCESSREAD = 0
CONST ACCESSWRITE = 1
CONST ACCESSREADWRITE = 2
CONST SHARECOMPAT = 0
CONST SHAREDENYREADWRITE = 1
CONST SHAREDENYWRITE = 2
CONST SHAREDENYREAD = 3
CONST SHAREDENYNONE = 4

COMMON SHARED TRUE, FALSE, NormAttr, RevAttr
COMMON SHARED NdxDir$, DocDir$, LstDir$, ASCEND, DESCEND
COMMON SHARED KeyEMS, KYInvertNdxEMS, KYInvertDatEMS, WeightEMS
COMMON SHARED SixteenK, ThirtyTwoK&

'$INCLUDE: '\\vadia\c-drive\USER\include\types.bi'

DECLARE FUNCTION ArraySize% (ULrow, ULcol, LRrow, LRcol)
DECLARE FUNCTION DosError%
DECLARE FUNCTION EmsError%
DECLARE FUNCTION EmsGetPFSeg%
DECLARE FUNCTION EmsPagesFree%
DECLARE FUNCTION EmsLoaded%
DECLARE FUNCTION EmsNumPages%
DECLARE FUNCTION Exist% (FileName$)
DECLARE FUNCTION FileSize& (FileName$)
DECLARE FUNCTION OneColor% (Fore%, Back%)
DECLARE FUNCTION QInstr&% (SPos%, Source$, Srch$)
DECLARE FUNCTION WhichError%

DECLARE SUB Array2EMS (SEG Element AS ANY, ElSize%, NumEls%, Handle%)
DECLARE SUB BCopyT (SEG FromEl AS ANY, SEG ToEl AS ANY, ElSize%, NumEls%)
DECLARE SUB Chime (Number%)
DECLARE SUB EmsAllocMem (NumPages%, Handle%)
DECLARE SUB EmsGet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsGet1El (SEG Value AS ANY, ElSize%, ElNum%, Handle%)
DECLARE SUB EmsMapMem (Handle%, PhysPage%, LogPage%)
DECLARE SUB EmsRelMem (Handle%)
DECLARE SUB EmsSet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsSet1El (SEG Value AS ANY, ElSize%, ElNum%, Handle%)
DECLARE SUB FClose (Handle%)
DECLARE SUB FGetA (Handle%, BYVAL SegAdr%, BYVAL Adr%, NumBytes&)
DECLARE SUB FGetAH (FileName$, SEG Address AS ANY, ElSize%, NumEls%)
DECLARE SUB FGetRT (Handle%, Dest AS ANY, RecNo&, RecSize%)
DECLARE SUB FOpen (FileName$, Handle%)
DECLARE SUB FOpenAll (FileName$, AccessMode%, ShareMode%, Handle%)
DECLARE SUB FSeek (Handle%, Location&)
DECLARE SUB InitInt (SEG Address%, StartValue%, NumEls%)
DECLARE SUB ISortS (SEG Element!, SEG IndexElement%, NumElements%, Direction%)
DECLARE SUB QPrintRC (Work$, Row%, Col%, Clr%)
DECLARE SUB SortT (SEG Array AS ANY, NumEls%, Direction%, ElSize%, MemOffset%, MemSize%)

'===========================================================

DECLARE SUB Config ()
DECLARE SUB DispMsg (Msg$, r%, c%)
DECLARE SUB ReleaseEMS ()
DECLARE SUB WindMgr (ULrow%, ULcol%, LRrow%, LRcol%, Frame%, BoxClr%, TextClr%, Tex$)
DECLARE FUNCTION EmsAlloc% (NumPages, Handle%, FileName$)
DECLARE FUNCTION LoadIntoEMS% (File$)

STACK 8192
```

```
FUNCTION FileDate$ (FInfo AS FileInfoType) STATIC

Year$ = MID$(STR$(FInfo.Year), 2)
Month$ = MID$(STR$(FInfo.Month), 2)
IF LEN(Month$) = 1 THEN Month$ = "0" + Month$
Day$ = MID$(STR$(FInfo.Day), 2)
IF LEN(Day$) = 1 THEN Day$ = "0" + Day$
Hour$ = MID$(STR$(FInfo.Hour), 2)
IF LEN(Hour$) = 1 THEN Hour$ = "0" + Hour$
Minute$ = MID$(STR$(FInfo.Minute), 2)
IF LEN(Minute$) = 1 THEN Minute$ = "0" + Minute$
Second$ = MID$(STR$(FInfo.Second), 2)
IF LEN(Second$) = 1 THEN Second$ = "0" + Second$

FileDate$ = Year$ + Month$ + Day$ + Hour$ + Minute$ + Second$

END FUNCTION

FUNCTION Unique$ (Path$)
        IF LEN(Path$) AND RIGHT$(Path$, 1) <> "\" THEN Path$ = Path$ + "\"
        Seed& = ABS(Peek2%(0, &H46C))           'use the TIMER as a seed
        DO
                TempName$ = Path$ + MID$(STR$(Seed&), 2) 'make a string out of it
                Seed& = Seed& + 1,                       'increment for next time
        LOOP UNTIL NOT Exist%(TempName$)         'loop and try another name

        Unique$ = TempName$                      'this is the function output
END FUNCTION

FUNCTION VMSAlloc (NumBytes&, HandleX) STATIC

'-- create a temporary but unique filename
TempPath$ = ENVIRON$("TEMP")
TempFile$ = Unique$(TempPath$) + ".VMS"

'-- create the file
FCreate TempFile$
IF DOSErrorX THEN
        VMSAlloc = FALSE
        VMSErrorX = 133
        EXIT FUNCTION
END IF

'-- open the file
FOpen TempFile$, HandleX
IF DOSErrorX THEN
        VMSAlloc = FALSE
        VMSErrorX = 133
        KILL TempFile$
        EXIT FUNCTION
END IF
'-- allocate disk space: seek, write something, close, reopen
QPrintRC "Allocating Space...", 24, 20, -1
FSeek HandleX, NumBytes& - 1
x$ = "X"
FPut HandleX, x$
FClose HandleX
QPrintRC "                    ", 24, 20, -1
FOpen TempFile$, HandleX
IF DOSErrorX THEN
        VMSAlloc = FALSE
        VMSErrorX = 136
        FClose HandleX
        KILL TempFile$
        EXIT FUNCTION
END IF

gVMSNumAllocations = gVMSNumAllocations + 1
REDIM gVMSTable(1 TO gVMSNumAllocations) AS VMSTableType

gVMSTable(gVMSNumAllocations).Handle = HandleX
gVMSTable(gVMSNumAllocations).TempFile = TempFile$

VMSAlloc = TRUE

END FUNCTION

FUNCTION VMSRelease (HandleX) STATIC

IF HandleX = 0 THEN
        VMSRelease = FALSE
        EXIT FUNCTION
END IF

'-- close the file
FClose HandleX

'-- get the filename from the VMSTable
Found = 0
FOR i = 1 TO gVMSNumAllocations
        IF gVMSTable(i).Handle = HandleX THEN
                Found = i
                EXIT FOR
        END IF
NEXT

'-- found it, so get the filename, delete the file and erase the entry
IF Found THEN
```

```
'$$$keyword-flag$$$ "Last Author: %v  Revision: %v  Date: %f"
CONST Version$ = "Last Author: TED  Revision: 11  Date: 11-Aug-92,14:10:06"

CALL Config

SpcLoc = INSTR(2, COMMAND$, " ")
IF SpcLoc = 0 THEN
        Start = 1
ELSE
        Cmd$ = MID$(COMMAND$, SpcLoc)
        Start = VAL(Cmd$)

END IF

'------------------------------------------------ K Y I N V E R T E D ----

'-- Load the KYInverted Index Data File

File$ = NdxDir$ + "KYINVERT.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
DIM NdxTemp AS NdxType
NdxLEN = LEN(NdxTemp)
TotKeys = FileSize&(File$) \ NdxLEN
REDIM InvertNdx(1 TO TotKeys) AS NdxType
CALL FGetAH(File$, SEG InvertNdx(1), NdxLEN, TotKeys)

'-- transfer it to EMS

CALL Array2EMS(SEG InvertNdx(1), NdxLEN, TotKeys, KYInvertNdxEMS)
IF EmsError% THEN
        PRINT "EMS Error"; EmsError%; "occurred."
        STOP
END IF

'-- erase array: it's in EMS

ERASE InvertNdx

'---- open KYInvert.DAT for random access

DIM RecTEMP AS KeyInfoLONG
RecLEN = LEN(RecTEMP)
File$ = NdxDir$ + "KYINVERT.DAT"
IF NOT Exist%(File$) THEN
        CLS
        Chime 2
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
FOpenAll File$, ACCESSREAD, SHAREDENYWRITE, KYInvertDatFILE

'==== Load WEIGHT.NDX into EMS if enough room, else open for random access

File$ = NdxDir$ + "WEIGHT.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
DIM WeightTEMP AS WeightNdx127
WeightLEN = LEN(WeightTEMP)
WeightEMS = LoadIntoEMS(File$)
IF WeightEMS = 0 THEN '-- not enough EMS to load it in, so just open it
        FOpenAll File$, ACCESSREAD, SHAREDENYWRITE, WeightFILE
END IF

'==== Load KEY.NDX into EMS if enough room, else open for random access

File$ = NdxDir$ + "KEY.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
DIM KeyTEMP AS KeyNdx127
KeyLEN = LEN(KeyTEMP)
KeyEMS = LoadIntoEMS(File$)
IF KeyEMS = 0 THEN '-- not enough EMS to load it in, so just open it
        FOpenAll File$, ACCESSREAD, SHAREDENYWRITE, KeyFILE
END IF

DIM Freq127 AS FreqComp127
DIM BlankFreq AS FreqComp127
Freq127LEN = LEN(Freq127)

'---- Open the FreqComp.127 file for writing (random access)
FreqFile$ = NdxDir$ + "FREQCOMP.127"
IF Exist%(FreqFile$) AND Start = 1 THEN KILL FreqFile$
OPEN FreqFile$ FOR RANDOM SHARED AS #1 LEN = Freq127LEN

'-- Index() is used for the pointer-based integer sort
REDIM Index(0 TO TotKeys - 1) AS INTEGER
```

```
FS$ = "FREQCOMP Started At Record" + STR$(Start) + "On " + DATE$ + " " + TIME$
QPrintRC FS$, 1, (80 - LEN(FS$)) \ 2, -1
QPrintRC Version$, 2, 7, -1

CALL QPrintRC("Processing 0     out of" + STR$(TotKeys), 10, 10, -1)

FOR Word = Start TO TotKeys

        CALL QPrintRC(STR$(Word), 10, 20, -1)

        EmsGet1El NdxTemp, NdxLEN, Word, KYInvertNdxEMS

        IF NdxTemp.Num THEN

                '-- clear the Value() for each Word
                REDIM Value(0 TO TotKeys) AS SINGLE

                FOR InvIndex& = NdxTemp.Index TO NdxTemp.Index + NdxTemp.Num - 1

                        '---- get the document # pointed to by KYINVERT.NDX.Index
                        '     from KYINVERT.DAT
                        CALL FGetRT(KYInvertDatFILE, RecTEMP, InvIndex&, RecLEN)

                        '---- get the key codes and number for this record
                        IF KeyEMS THEN
                                EmsGet KeyTEMP, KeyLEN, RecTEMP.Rec, KeyEMS
                        ELSE
                                FGetRT KeyFILE, KeyTEMP, RecTEMP.Rec, KeyLEN
                        END IF

                        '---- get the Weight values for this record
                        IF WeightEMS THEN
                                EmsGet WeightTEMP, WeightLEN, RecTEMP.Rec, WeightEMS
                        ELSE
                                FGetRT WeightFILE, WeightTEMP, RecTEMP.Rec, WeightLEN
                        END IF

                        FOR Keyword = 1 TO KeyTEMP.Num

                                Code = KeyTEMP.Code(Keyword)

                                '---- add Weight to Code's total value sum
                                IF WeightTEMP.Weight(Keyword) = 0 THEN STOP
                                Value(Code) = Value(Code) + WeightTEMP.Weight(Keyword)

                        NEXT' keyword in this record

                NEXT 'next record that Word appears in

                Freq127.Code = Word

                '---- scale the values to the Freq127 word itself
                '     use the code number to obtain it's value
                ScaleValue! = Value(Freq127.Code)

                '---- set the value to zero so it won't come up after sorting
                Value(Freq127.Code) = 0

                '---- pointer sort in decreasing order
                CALL InitInt(SEG Index(0), 1, TotKeys)
                CALL ISortS(SEG Value(0), SEG Index(0), TotKeys, DESCEND)

                FOR i = 1 TO 127' max number of Freq Comps

                        Value! = Value(Index(i - 1))

                        IF Value! > 0 THEN ' store it

                                Freq127.Comp(i) = Index(i - 1)
                                V% = Value! / ScaleValue! * 100
                                IF V% > 255 THEN V% = 255
                                Freq127.Value(i) = CHR$(V%)

                        ELSE

                                EXIT FOR

                        END IF

                NEXT

                Freq127.Num = i - 1' 127 if loop completed, else where it stopped

                PUT #1, Word, Freq127

        ELSE

                PUT #1, Word, BlankFreq

        END IF

        IF INKEY$ = CHR$(27) THEN
                Chime 5
                PRINT "Press ENTER to abort."
                DO
                        i$ = INKEY$
                LOOP UNTIL LEN(i$)
                IF i$ = CHR$(13) THEN EXIT FOR
                CLS
        END IF
```

```
NEXT' next Word to process

CLOSE #1' close the freq coop 127 list

CALL ReleaseEMS

SUB Config STATIC

IF COMMAND$ <> "" THEN
        ConfigFile$ = COMMAND$
        SpcLoc = INSTR(ConfigFile$, " ")
        IF SpcLoc THEN ConfigFile$ = LEFT$(ConfigFile$, SpcLoc - 1)
        ConfigFile$ = ConfigFile$ + ".CFG"
        PRINT "Reading "; ConfigFile$
ELSE
        Chime 5
        PRINT "No configuration file was given."
        PRINT "Press any key to continue."
        DO: LOOP UNTIL LEN(INKEY$)
END IF
IF NOT Exist(ConfigFile$) THEN
        Chime 10
        PRINT "File"; ConfigFile$; "was not found."
        PRINT "Press any key to continue."
        DO: LOOP UNTIL LEN(INKEY$)
END IF

OPEN ConfigFile$ FOR INPUT AS #1
        INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$, x!
CLOSE #1

COLOR Fg, Bg, Brdr

NormAttr = OneColor%(Fg, Bg)
RevAttr = OneColor%(Bg, Fg AND 7)

SixteenK = 16384
ThirtyTwoK& = SixteenK * 2&

CLS
IF NOT EmsLoaded% THEN
        Chime 8
        PRINT "No EMS loaded."
        STOP
END IF

FALSE = 0
TRUE = NOT FALSE
ASCEND = 0
DESCEND = NOT ASCEND

END SUB

SUB DispMsg (Msg$, r, c) STATIC
STATIC WindOpen, Scr%() ' is there already a message displayed?
SHARED Fg, Bg

IF LEN(Msg$) = 0 AND WindOpen THEN GOSUB MsgClose: EXIT SUB
IF WindOpen THEN
        CALL Chime(9)
        OPEN "DEBUG" FOR OUTPUT AS #9
        PRINT #9, "WindOpen="; WindOpen; HEX$(WindOpen)
        PRINT #9, "Msg$= |"; Msg$; "|"
        PRINT #9, "TRUE="; TRUE; HEX$(TRUE)
        PRINT #9, "FALSE="; FALSE; HEX$(FALSE)
        CLOSE #9
        CALL Chime(8)
        CLS
        WHILE LEN(INKEY$) = 0: WEND
        GOSUB MsgClose
END IF

Wid = LEN(Msg$)
IF Wid > 50 THEN Wid = 50

Msg$ = Msg$ + " "' make sure there's a space to find at the end (see below)

MaxLin = LEN(Msg$) \ Wid + 3
IF MaxLin > 23 THEN MaxLin = 23
REDIM Text$(MaxLin)
Lin = 0
DO
        Lin = Lin + 1' incr current lin # (also element in text display array)
        lastspc = QInstrRC(Wid + 1, Msg$, " ")' look for the last space so we can word wrap
        Text$(Lin) = LEFT$(Msg$, lastspc - 1)
        Msg$ = MID$(Msg$, lastspc + 1)'remove portion of string that's in t$
LOOP WHILE LEN(Msg$) > Wid

Msg$ = RTRIM$(Msg$)
IF LEN(Msg$) THEN
        Lin = Lin + 1
        Text$(Lin) = Msg$
END IF

IF r <> 0 AND c = 0 THEN
        ULr = r
ELSE
        ULr = 9
END IF
        DULr = ULr - 1
        LRr = ULr + Lin + 1
        DLRr = LRr + 2
```

```
horizmargin = (80 - Wid) \ 2
        ULc = horizmargin
        DULc = ULc - 3
        LRc = 80 - horizmargin
        IF (Wid \ 2) * 2 = Wid THEN LRc = LRc + 1' if it's even WIDth, then bump up the LRcolumn.
        DLRc = LRc + 1

REDIM Scr%(ArraySize%(DULr, DULc, DLRr, DLRc))
CALL ScrnSaveO(DULr, DULc, DLRr, DLRc, SEG Scr%(0))
CALL WindMgr(ULr, ULc, LRr, LRc, 4, NormAttr, RevAttr, "Status")
FOR i = 1 TO Lin
        CALL QPrintRC(Text$(i), ULr + 1, ULc + 1, -1)
NEXT

r = ULr + Lin
c = ULc + 1 + LEN(Text$(Lin))
IF LEN(Text$(Lin)) + 2 = Wid THEN c = ULc + 1: r = r + 1

ERASE Text$
WindOpen = TRUE

EXIT SUB

'------------------------------------------------------------ close window
MsgClose:

CALL ScrnRestO(DULr, DULc, DLRr, DLRc, SEG Scr%(0))

ERASE Scr%
WindOpen = FALSE
RETURN

END SUB

FUNCTION EmsAlloc% (NumPgs%, Handle%, FileName$) STATIC

IF EmsPagesFree% < NumPgs THEN
        EmsAlloc = FALSE
ELSE
        EmsAllocMem NumPgs%, Handle%
        EmsAlloc = TRUE
END IF

END FUNCTION

FUNCTION LoadIntoEMS (File$) STATIC

'--------- Returns the handle where the file was loaded into ---------

EMSPg = EmsGetPFSeg%

SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
IF NOT EmsAlloc(NumPages, FileEMS, File$) THEN
        LoadIntoEMS = 0
        EXIT FUNCTION
END IF

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)

FOpen File$, LoadFile
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1) + "      ", 4, 1, -1

        '-- map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT

        '-- seek to beginning of current block
        FSeek LoadFile, (i - 1) * ThirtyTwoK&
        IF DosError% THEN PRINT "Dos Error:"; WhichError%: STOP

        IF i < Num32kBlocks + 1 THEN
                '-- get the 32k block and put it directly into the EMS page frame
                FGetA LoadFile, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&
                IF DosError% THEN PRINT "Dos Error:"; WhichError%: STOP
        ELSE
                '-- load the left over (<32k) bytes
                FGetA LoadFile, BYVAL EMSPg, BYVAL 0, LeftOver&
                IF DosError% THEN PRINT "Dos Error:"; WhichError%: STOP
        END IF

NEXT

FClose LoadFile

LoadIntoEMS = FileEMS

END FUNCTION

SUB ReleaseEMS STATIC

IF KeyEMS THEN
        EmsReUMem KeyEMS
ELSEIF KeyFILE THEN
        FClose KeyFILE
END IF
```

```
IF KYInvertNdxEMS THEN EmsRelMem KYInvertNdxEMS

IF KYInvertDatEMS THEN EmsRelMem KYInvertDatEMS

IF WeightEMS THEN

        EmsRelMem WeightEMS
ELSEIF WeightFILE THEN
        FClose WeightFILE
END IF

END SUB

SUB WindMgr (ULrow, ULcol, LRrow, LRcol, Frame, BoxColr, TextColr, Text$) STATIC

CALL BoxO(ULrow - 1, ULcol - 1, LRrow + 1, LRcol + 1, Frame, BoxColr)
CALL ClearScrO(ULrow, ULcol, LRrow, LRcol, BoxColr)
CALL QPrintRC("[" + Text$ + "]", ULrow - 1, ULcol + 1, TextColr)

END SUB

DEFINT A-Z

'==================
'   RELATIVE.BAS
'==================

'---- are there any FreqComps for this keyword?
'---- we found the word in j's FCList
'---- look for the word itself in word j's FCList
'---- apply formula of (Lower * 6 + Higher)/7
'---- sort in decreasing order, by value
'---- save first 63 as the relatives

'Invoked: RELATIVE ConfigFile
'--------

'Creates: Relative.63
'--------
'  nth Record of Relative.63 contains nth Code
'  Num of relatives of this code &


'Uses: FreqComp.127, Dict.Wrd (for NumKeys keyword count)
'-----

'63 paired values: Companion Codes (Comp()) and 100 * Weight (Value())
'  TYPE FreqComp63
'     Num AS INTEGER
'     Code AS INTEGER
'     Comp(1 TO 63) AS INTEGER
'     Value(1 TO 63) AS STRING * 1

'$INCLUDE: '\\vadim\c-drive\user\include\types.bi'

' Color Attributes
COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr

' Directories
COMMON SHARED LstDir$, DocDir$, KeyDir$, NdxDir$

'$INCLUDE: '\\vadim\c-drive\user\include\const.bi'

'------------------------------------------------------------ Internal SUBs

DECLARE SUB Config ()
DECLARE SUB EmsAlloc (NumPgs%, Handle%)

'------------------------------------------------------------ External SUBs

DECLARE SUB Chime (Number)
DECLARE SUB EmsAllocMem (NumPages%, Handle%)
DECLARE SUB EmsMapMem (Handle%, PhysicalPage%, LogicalPage%)
DECLARE SUB EmsGet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsSet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsRelMem (Handle%)
DECLARE SUB FClose (Handle%)
DECLARE SUB FCreate (FileName$)
DECLARE SUB FGetA (Handle%, BYVAL SegAdr%, BYVAL Adr%, NumBytes&)
DECLARE SUB FGetAH (FileName$, SEG Element AS ANY, ElSize%, NumEls%)
DECLARE SUB FGetRT (Handle%, Destination AS ANY, RecNumber&, RecSize%)
DECLARE SUB FOpen (FileName$, Handle%)
DECLARE SUB FPutT (Handle%, Source AS ANY, RecSize%)
DECLARE SUB FPutRT (Handle%, Source AS ANY, RecNumber&, RecSize%)
DECLARE SUB FSeek (Handle%, Location&)
DECLARE SUB QPrintRC (St$, Row%, Col%, Colr%)

DECLARE SUB SortT (SEG Address AS ANY, NumEls%, Dir%, ElSize%, MemOffset%, MemSize%)

'---------------------------------------------------------- External FUNCTIONs

DECLARE FUNCTION EmsLoaded%
DECLARE FUNCTION EmsError%
DECLARE FUNCTION EmsPagesFree%
DECLARE FUNCTION EmsGetPFSeg%
DECLARE FUNCTION Exist% (FileName$)
DECLARE FUNCTION OneColor% (Fore%, Back%)
DECLARE FUNCTION FileSize& (FileName$)
DECLARE FUNCTION WhichError%
DECLARE FUNCTION DosError%
DECLARE FUNCTION MaxInt% (a%, b%)
DECLARE FUNCTION MinInt% (a%, b%)
```

```
'------------------------------------------------------ PROGRAM START

CALL Config

DIM NonMutualCount AS LONG, MutualCount AS LONG

Threshold = 0

ThirtyTwoK& = 32 * 1024&

EmsPg = EmsGetPFSeg%

'-- determine number of dictionary entries
File$ = LstDir$ + "DICT.WRD"
DIM DictEntry AS DictType
NumKeys = FileSize&(File$) \ LEN(DictEntry)

DIM Freq127 AS FreqComp127, Freq127obj AS FreqComp127
Freq127LEN = LEN(Freq127)

File$ = NdxDir$ + "FREQCOMP.127"
SizeofFile& = FileSize&(File$)
NumFreqComp = SizeofFile& \ Freq127LEN

NumPages = SizeofFile& \ 16384 + 2' round off to nearest " "ages
EmsAlloc NumPages, FreqCompEMS
IF EmsError% THEN PRINT "Ems error:"; EmsError%

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)

FOpen File$, FreqCompFILE
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC "Loading block" + STR$(i) + " /" + STR$(Num32kBlocks + 1), 3, 1, -1

        '-- map pages of the FreqCompEMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FreqCompEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT

        '-- seek to beginning of current block
        FSeek FreqCompFILE, (i - 1) * ThirtyTwoK&
        IF DosError% THEN PRINT "Dos Error:"; WhichError%

        IF i < Num32kBlocks + 1 THEN
                '-- get the 32k block and put it directly into the EMS page frame
                FGetA FreqCompFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&
                IF DosError% THEN PRINT "Dos Error:"; WhichError%
        ELSE
                '-- load the left over (<32k) bytes
                FGetA FreqCompFILE, BYVAL EmsPg, BYVAL 0, LeftOver&
                IF DosError% THEN PRINT "Dos Error:"; WhichError%
        END IF

NEXT

FClose FreqCompFILE

CLS

DIM Rel AS FreqComp63, BlankRel AS FreqComp63
RelLEN = LEN(Rel)

'---- Initialize the Blank Relative structure
BlankRel.Num = 0: BlankRel.Code = 0
FOR i = 1 TO 63
        BlankRel.Comp(i) = 0
        BlankRel.Value(j) = CHR$(0)
NEXT

File$ = NdxDir$ + "RELATIVE.63"
FCreate File$
FOpen File$, RelativeFile

CLS

QPrintRC "Processing Keyword 0     out of " + STR$(NumKeys), 10, 10, -1

FOR i = 1 TO NumFreqComp

        EmsGet Freq127, Freq127LEN, CLNG(i), FreqCompEMS

        ' fill this rec with 0's in case there are no FreqComps
        ' also to clear the variable in case there aren't 63 Relatives

        Rel = BlankRel

        Rel.Code = Freq127.Code

        QPrintRC STR$(i), 10, 29, -1

        '---- are there any FreqComps for this keyword?

        IF Freq127.Num THEN '-- yes, so process it

             WordsInUse = WordsInUse + 1

             REDIM RelVal(1 TO Freq127.Num) AS CodeValue
```

```
        FOR j = 1 TO Freq127.Num

                RelVal(j).Code = Freq127.Comp(j)
                RelVal(j).Value = ASC(Freq127.Value(j))

                EmsGet Freq127obj, Freq127LEN, CLNG(Freq127.Comp(j)), FreqCompEMS

                Found = FALSE

                '-- look for the word itself in word j's FCList
                FOR k = 1 TO Freq127obj.Num

                        IF Freq127obj.Comp(k) = Freq127.Code THEN

                                '-- we found the word in j's FCList
                                '   apply formula of Lower * 6 + Higher
                                '                    ------------------
                                '                           7

                                ObjValue = ASC(Freq127obj.Value(k))

                                Max = MaxInt(RelVal(j).Value, ObjValue)
                                Min = MinInt(RelVal(j).Value, ObjValue)

                                RelVal(j).Value = (Max + (Min * 6)) / 7

                                Found = TRUE

                                EXIT FOR

                        END IF

                NEXT

                IF Found THEN  '- mutual
                        MutualCount = MutualCount + 1
                ELSE  '- non-mutual
                        NonMutualCount = NonMutualCount + 1
                        RelVal(j).Value = RelVal(j).Value / 8
                END IF

        NEXT

                '---- sort in decreasing order, by value

                SortT RelVal(1), Freq127.Num, DESCEND, LEN(RelVal(1)), LEN(RelVal(1).Code), -1

                '---- save first 63 as the relatives

                MaxRel = 63
                IF MaxRel > Freq127.Num THEN MaxRel = Freq127.Num

                FOR j = 1 TO MaxRel

                        IF RelVal(j).Value >= ThresHold THEN

                                Rel.Comp(j) = RelVal(j).Code
                                Rel.Value(j) = CHR$(RelVal(j).Value)
                                Rel.Num = Rel.Num + 1

                        ELSE

                                EXIT FOR

                        END IF

                NEXT

        END IF

        '---- store this record on disk

        FPutRT RelativeFile, Rel, CLNG(i), RelLEN

        IF INKEY$ = CHR$(27) THEN Chime 6: EXIT FOR

NEXT

' close output file
FClose RelativeFile

' release FreqComp use of EMS
EmsRelMem FreqCompEMS

OPEN "MUTUAL.DAT" FOR OUTPUT AS 1
        PRINT #1, "Number of Mutual:      "; MutualCount
        PRINT #1, "Number of Non-Mutual:  "; NonMutual.Count
        PRINT #1, "Number of Words in use:"; WordsInUse
CLOSE #1

SUB Config STATIC

IF COMMAND$ <> "" THEN
        ConfigFile$ = COMMAND$ + ".CFG"
ELSE
        PRINT "No configuration file was given."
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF
```

```
IF ExistZ%(ConfigFile$) THEN
        OPEN ConfigFile$ FOR INPUT AS #1
                INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
        CLOSE #1
ELSE
        CALL Chime(6)
        PRINT "Configuration file "; ConfigFile$; " Does not exist."
        PRINT "Press any key to continue."
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF

COLOR Fg, Bg, Brdr

NormAttr = OneColor%(Fg, Bg)
RevAttr = OneColor%(Bg, Fg AND 7)

CLS
QPrintRC "RELATIVE", 1, 36, -1
IF NOT EmsLoaded% THEN
        PRINT "No EMS!"
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF

END SUB

SUB EmsAlloc (NumPgs%, Handle%) STATIC

IF EmsPagesFree% < NumPgs THEN
        CALL Chime(8)
        PRINT "Couldn't Allocate "; CLNG(NumPgs) * Sixteen&; " bytes of EMS ("; NumPgs; " pages)."
        PRINT "Only "; EmsPagesFree%; " pages were available."
        STOP
ELSE
        EmsAllocMem NumPgs%, Handle%
END IF

END SUB

DEFINT A-Z

'REL-INV.BAS
'Invoked: rel-inv ConfigFile
'Creates: Rel-Inv.Ndx, Rel_Inv.Dat
'Uses: Relative.63 & Dict.Vrd for NumKeys
'nth Rec of Rel-Inv.Ndx contains nth Code, ptr into Rel-Inv.Dat & NumRelsOfCode
'  TYPE NdxType
'     Code AS INTEGER
'     Index AS LONG
'     Num AS INTEGER
'pointed to record of Rel-Inv.Dat contains code of the first FreqComp list
'of nth code & the nth code's value in that list
'  TYPE RecValue
'     Rec AS INTEGER
'     Value AS STING * 1

'$INCLUDE: '\\vadim\c-drive\user\include\types.bi'

TYPE LinkHead
        Num AS INTEGER
        FirstPtr AS LONG
        LastPtr AS LONG
        Pad AS STRING * 6
END TYPE

TYPE LinkNode
        ' holds the actual data, in this case the FC record # and its value
        Info AS RecValue
        Ptr AS LONG
END TYPE


' Color Attributes
COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr

' Directories
COMMON SHARED LstDir$, DocDir$, KeyDir$, NdxDir$

' Temp variables
COMMON SHARED NodeLEN%, NodeTemp AS LinkNode

'$INCLUDE: '\\vadim\c-drive\user\include\const.bi'

'------------------------------------------------------------ Internal SUBs

DECLARE SUB AddNode (Head AS ANY, Info AS RecValue, StorageH AS INTEGER, FreePtr AS LONG)
DECLARE SUB Config ()
DECLARE SUB EmsAlloc (NumPgs%, Handle%)

'------------------------------------------------------------ External SUBs

DECLARE SUB EmsAllocMem (NumPages%, Handle%)
DECLARE SUB EmsGet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsSet (SEG Value AS ANY, ElSize%, ElNum&, Handle%)
DECLARE SUB EmsRelMem (Handle%)
DECLARE SUB FClose (Handle%)
DECLARE SUB FCreate (FileName$)
DECLARE SUB FGetRT (Handle%, Destination AS ANY, RecNumber&, RecSize%)
DECLARE SUB FOpen (FileName$, Handle%)
DECLARE SUB FPutAH (FileName$, SEG Element AS ANY, ElSize%, NumEls%)
```

```
DECLARE SUB FPutT (HandleX, Source AS ANY, RecSizeX)
DECLARE SUB QPrintRC (St$, RowX, ColX, ColrX)

'$---------------------------------------------------------- External FUNCTIONs

DECLARE FUNCTION EmsLoadedX
DECLARE FUNCTION EmsPagesFreeX
DECLARE FUNCTION ExistX (FileName$)
DECLARE FUNCTION OneColorX (ForeX, BackX)
DECLARE FUNCTION FileSize& (FileName$)

'$---------------------------------------------------------- PROGRAM START

CALL Config

LoadFile$ = LstDir$ + "DICT.VRD"
DIM d AS DictType
NumKeys = FileSize&(LoadFile$) \ LEN(d)

DIM Freq63 AS FreqComp63
Freq63LEN = LEN(Freq63)

LoadFile$ = NdxDir$ + "RELATIVE.63"
NumFreqComp = FileSize&(LoadFile$) \ Freq63LEN
FOpen LoadFile$, FreqCompH%

REDIM LinkList(1 TO NumKeys) AS LinkHead

DIM Info AS RecValue
InfoLEN = LEN(Info)

NumPages = NodeLEN * CLNG(NumFreqComp) * 63 \ 16384& + 1
EmsAlloc NumPages, InvertDatEMS

FreePtr& = 1

CLS

FOR i = 1 TO NumFreqComp

        QPrintRC STR$(i) + " out of " + STR$(NumFreqComp), 10, 1, -1

        FGetRT FreqCompH%, Freq63, CLNG(i), Freq63LEN

        Info.Rec = Freq63.Code

        FOR j = 1 TO Freq63.Num

                Info.Value = Freq63.Value(j)

                AddNode LinkList(Freq63.Comp(j)), Info, InvertDatEMS, FreePtr&

        NEXT

NEXT

' close input file
FClose FreqCompH%

' ---------------------------------------------------------
' start writing inverted file by traversing the linked list
' ---------------------------------------------------------

CLS

REDIM Ndx(1 TO NumKeys) AS NdxType

FCreate NdxDir$ + "REL-INV.DAT"
FOpen NdxDir$ + "REL-INV.DAT", InvertDatFile

Index& = 1

FOR i = 1 TO NumKeys

        QPrintRC STR$(i) + " " + STR$(LinkList(i).Num) + "  ", 10, 1, -1

        Ptr& = LinkList(i).FirstPtr

        Ndx(i).Index = Index&
        Ndx(i).Num = LinkList(i).Num

        DO WHILE Ptr&

                EmsGet NodeTemp, NodeLEN, Ptr&, InvertDatEMS
                FPutT InvertDatFile, NodeTemp.Info, InfoLEN
                Ptr& = NodeTemp.Ptr

        LOOP

        Index& = Index& + Ndx(i).Num

NEXT

EmsRelMem InvertDatEMS

FClose InvertDatFile

FPutAH NdxDir$ + "REL-INV.NDX", Ndx(1), LEN(Ndx(1)), NumKeys

DATA "   Copyright 1990 by Ted M. Young.  ALL RIGHTS RESERVED.   "
```

```
SUB AddNode (Head AS LinkHead, Info AS RecValue, StorageH AS INTEGER, FreePtr AS LONG) STATIC

IF Head.FirstPtr = NULL THEN

        Head.FirstPtr = FreePtr
        Head.LastPtr = FreePtr

ELSE

        CALL EmsGet(NodeTemp, NodeLEN, Head.LastPtr, StorageH)
        NodeTemp.Ptr = FreePtr
        CALL EmsSet(NodeTemp, NodeLEN, Head.LastPtr, StorageH)

        Head.LastPtr = FreePtr

END IF

Head.Num = Head.Num + 1

NodeTemp.Info = Info
NodeTemp.Ptr = NULL
CALL EmsSet(NodeTemp, NodeLEN, FreePtr, StorageH)

FreePtr = FreePtr + 1

END SUB

SUB Config STATIC

IF COMMAND$ <> "" THEN
        ConfigFile$ = COMMAND$ + ".CFG"
ELSE
        PRINT "No configuration file was given."
        PRINT "Press any key to continue."
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF

IF ExistX(ConfigFile$) THEN
        OPEN ConfigFile$ FOR INPUT AS #1
                INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$
        CLOSE #1
ELSE
        CALL Chime(6)
        PRINT "Configuration file "; ConfigFile$; " does not exist."
        PRINT "Press any key to continue."
        DO: LOOP UNTIL LEN(INKEY$)
        END
        END
END IF

COLOR Fg, Bg, Brdr

NormAttr = OneColorX(Fg, Bg)
RevAttr = OneColorX(Bg, Fg AND 7)

CLS
QPrintRC "REL-INV", 1, 36, -1
IF NOT EmsLoadedX THEN
        PRINT "No EMS!"
        CALL Chime(6)
        DO: LOOP UNTIL LEN(INKEY$)
        END
END IF

NodeLEN = LEN(NodeTemp)

END SUB

SUB EmsAlloc (NumPgsX, HandleX) STATIC

IF EmsPagesFreeX < NumPgs THEN
        CALL Chime(8)
        PRINT "Couldn't Allocate "; CLNG(NumPgs) * SixteenK; " bytes of EMS ("; NumPgs; " pages)."
        PRINT "Only "; EmsPagesFreeX; " pages were available."
        STOP
ELSE
        EmsAllocMax NumPgsX, HandleX
END IF

END SUB

DEFINT A-Z

'POLYSEMY.BAS: Builds: PolySemy.Dat & PolyAvg
'              Uses  : Relative.63, Rel-InvS.Ndx, Rel-Inv.Dat
'nth Rec of PolySemy.Dat contains Poly Value of nth word calculated as follows:

'POLY FORMULA:
'-------------
' PolySemy(Word) = LOG2( (1/600) * ( (Avg3 * (Avg3/Avg20) ) ^ 1.68 + (Avg6 * (Avg6/Avg63) ) ^ 1.8) * (5 ^ (NumWordsWhichAreRels/DocsWi

'New formula 5/28/92 EPG
' PolySemy(Word) = LOG2( (1/1000) * ( (Avg3 * (Avg3/Avg20) ) ^ 2.2 + (Avg6 * (Avg6/Avg63) ) ^ 2.2) * (4 ^ (NumWordsWhichAreRels/DocsWi
'       Equivalent formula=
' PolySemy(Word) = LOG2((Avg3*(Avg3/Avg20))^2.2+(Avg6*(Avg6/Avg63))^2.2)+LOG2(4)*(NumWordsWhichAreRels/DocsWithWord)-LOG2(1000)
' PolyAvg is average poly value

'New formula 6/1/92 VN
' PolySemy(Word) =(LOG2((Avg3*(Avg3/Avg20))^2.2+(Avg6*(Avg6/Avg63))^2.2)+1.8*(NumWordsWhichAreRels/DocsWithWord)-9))*(DocsWithWord^(1/
```

```
'== New formula 6/19/92 TMY
'                                                    /      Avg3        Avg6  \
' PolySemy(Word) = SQR( TotalRelValue / DocFreq ) x SQR( Avg3 * ------ + Avg6 * ------ )
'                                                    \      Avg20       Avg63 /
'
'    TotRelVal is the sum of relative values from all relative lists
'    in which the word appears

'$INCLUDE: '\\vadim\c-drive\user\include\types.bi'

TYPE PolyType
        Code AS INTEGER
        Value AS SINGLE
        Pad AS STRING * 2 '-- needed to make this type a ^2 for huge array
END TYPE

'-- Color Attributes
COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr

'-- Common Temp Variables
COMMON SHARED Temp$, i%, j%, k%, l%, x%

'-- Common Constants
COMMON SHARED SixteenK, ThirtyTwoK&

'-- File Directories
COMMON SHARED LstDir$, DocDir$, KeyDir$, NdxDir$
COMMON SHARED DictWordEMS, RelInvDatEMS

'$INCLUDE: '\\vadim\c-drive\user\include\const.bi'

'------------------------------------------------------ Internal SUBs

DECLARE SUB Config ()
DECLARE SUB DispMsg (Msg$, r%, c%)
DECLARE SUB EmsAlloc (NumPgs%, Handle%, FileName$)
DECLARE SUB ReleaseEMS ()
DECLARE SUB WindMgr (ULRow%, ULCol%, LRRow%, LRCol%, Frame%, BoxColr%, TextColr%, Text$)

'------------------------------------------------------ Internal FUNCTIONs

DECLARE FUNCTION Dict$ (Code%)
DECLARE FUNCTION Freq% (Array() AS NdxType, Tot%, Code%)
DECLARE FUNCTION LoadIntoEMS% (File$)

'------------------------------------------------------ External Declares

'$INCLUDE: '\\vadim\c-drive\user\include\declares.bi'

'------------------------------------------------------ PROGRAM START

CALL Config
QPrintRC "POLYSEMY", 1, 35, -1

'$$$keyword-flag$$$ "Last Author: %w  Revision: %v  Date: %f"
CONST Version$ = "Last Author: TED  Revision: 17  Date: 21-Aug-92,15:32:18"

Col = (80 - LEN(Version$)) \ 2
QPrintRC Version$, 2, Col, -1

'-- Load in Code-->Word dictionary directly into EMS

LoadFILE$ = LstDir$ + "DICT.WRD"
DictWordEMS = LoadIntoEMS(LoadFILE$)

'-- Load in REL-INV.DAT

LoadFILE$ = NdxDir$ + "REL-INV.DAT"
RelInvDatEMS = LoadIntoEMS(LoadFILE$)
DIM RelInvDat AS RecValue
RelInvDatLEN = LEN(RelInvDat)

'-- keyword inverted file index (which holds the keyword code and
'       the number of documents in which it appears [DocFreq])
LoadFILE$ = NdxDir$ + "KYINVERT.NDX"
DIM NdxTemp AS NdxType

'-- get the number of keywords
IF Exist(LoadFILE$) THEN
        NumKey = FileSize&(LoadFILE$) \ LEN(NdxTemp)
ELSE
        PRINT LoadFILE$; " was not found."
        ReleaseEMS
        END
END IF

'-- dimension array to hold the inverted index info
REDIM Ndx(1 TO NumKey) AS NdxType

'-- load the file into the array
CALL FGetAH(LoadFILE$, Ndx(1), LEN(NdxTemp), NumKey)

DIM Rel AS FreqComp63
RelLEN = LEN(Rel)

LoadFILE$ = NdxDir$ + "RELATIVE.63"
NumRel = FileSize&(LoadFILE$) \ RelLEN
CALL FOpen(LoadFILE$, RelativeFile)

'** DIM RelInvNdx AS NdxType
```

```
REDIM RelInvNdx(1 TO NumRel) AS SmallNdxType
LoadFILE$ = NdxDir$ + "REL~INVs.NDX"

'** CALL FOpen(LoadFILE$, RelInvNdxFile)

'-- load the file into the array
FGetAH LoadFILE$, RelInvNdx(1), LEN(RelInvNdx(1)), NumRel

CLS

PRINT "Processing"; NumRel; " words..."

QPrintRC "First Pass Polyvalues:", 10, 1, -1
QPrintRC "Second Pass, Final Polyvalues:", 11, 1, -1
QPrintRC "Printing Pass, writing to disk:", 12, 1, -1

REDIM PolySemy(1 TO NumRel) AS PolyType
REDIM PolySemyOrig(1 TO NumRel) AS SINGLE
REDIM TotRelValue(1 TO NumRel) AS INTEGER

FOR i = 1 TO NumRel

        CALL QPrintRC(STR$(i), 10, 23, -1)

        CALL FGetRT(RelativeFile, Rel, CLNG(i), RelLEN)

        PolySemy(i).Code = i
        PolySemy(i).Value = 0
        PolySemyOrig(i) = 0

        IF Rel.Num > 0 THEN ' calculate it, otherwise skip it

                Sum = 0

                Avg3! = 0: Avg6! = 0: Avg20! = 0: Avg63! = 0

                FOR j = 1 TO 63

                        IF j <= Rel.Num THEN
                                Code = Rel.Comp(j)
                                '** Fq = Ndx(Code).Num '** Not used?
                                Value = ASC(Rel.Value(j))
                        ELSE
                                Value = 0
                        END IF

                        Sum = Sum + Value
                        SELECT CASE j
                                CASE 3
                                        Avg3! = Sum / 3
                                CASE 6
                                        Avg6! = Sum / 6
                                CASE 20
                                        Avg20! = Sum / 20
                                CASE 63
                                        Avg63! = Sum / 63
                                CASE ELSE
                        END SELECT

                NEXT

                IF Avg63! > 0 AND Avg20! > 0 THEN
                        '-- SQR is the same as ^ .5
                        Poly! = SQR(Avg3! * (Avg3! / Avg20!) + Avg6! * (Avg6! / Avg63!))
                ELSE
                        Poly! = 0
                END IF

                PolySemy(i).Value = Poly!
                PolySemyOrig(i) = Poly!

        END IF

        IF INKEY$ = CHR$(27) THEN
                INPUT "Exit?", yn$
                IF UCASE$(LEFT$(yn$, 1)) = "Y" THEN EXIT FOR
        END IF

NEXT

CALL FClose(RelativeFile)
'=============================================================

DIM PolyValue AS PolyValueType

OPEN NdxDir$ + "POLYSEMY.DAT" FOR RANDOM AS 1 LEN = LEN(PolyValue)

Log2! = LOG(2)
Log4! = LOG(4)
Log1000! = LOG(1000)

AvgPoly! = 0

FOR i = 1 TO NumRel

        CALL QPrintRC(STR$(i), 11, 31, -1)

        IF PolySemy(i).Value > 0 THEN

                '** FGetRT RelInvNdxFile, RelInvNdx, CLNG(i), LEN(RelInvNdx)
                Polyfreq = Ndx(i).Num
```

```
'               PolyValue.Value = PolySemy(i).Value * ((RelInvNdx.Num / PolyFreq) ^ 4)
'     changed 3/12/92 VN
'               PolyValue.Value = PolySemy(i).Value / 20 * (5 ^ (RelInvNdx.Num / PolyFreq))
'     changed 3/17/92 VN
'               PolyValue.Value = LOG(PolySemy(i).Value)
'               IF RelInvNdx.Num > 0 THEN

                PolyTemp! = RelInvNdx(i).Num / PolyFreq

'               ELSE
'                  polytemp! = 0
'               END IF
'               PolyValue.Value = PolyValue.Value + Log4! * polytemp! - Log1000!
'               PolyValue.Value = PolyValue.Value / Log2!
'**   Changed 6/01/92 VN
'**       IF RelInvNdx(i).Num > 0 THEN
'**          PolyTemp! = (RelInvNdx(i).Num / PolyFreq) ^ .7
'**       ELSE
'**          PolyTemp! = .1
'**       END IF

                '=== new calculation method 8/19/92 - TMY
                '-- TotRelVal is the sum of relative values from all relative lists
                '   in which the word appears
                TotRelVal& = 0
                FOR j& = RelInvNdx(i).Index TO RelInvNdx(i).Index + RelInvNdx(i).Num - 1
                        EmsGet RelInvDat, RelInvDatLen, j&, RelInvDatEMS
                        TotRelVal& = TotRelVal& + ASC(RelInvDat.Value)
                NEXT

                Temp& = TotRelVal& AND &HFFFF&
                IF Temp& < 32768 THEN
                        TotRelValue(i) = Temp&
                ELSE
                        TotRelValue(i) = Temp& - 65536
                END IF
                IF TotRelVal& > 0 THEN
                        '-- changed 8/21/92: PolyFreq is now PolyFreq ^ 1.2
                        PolyTemp! = SQR(TotRelVal& / PolyFreq ^ 1.2)
                ELSE
                        PolyTemp! = .1
                END IF
                PolyValue.Value = PolyTemp! * PolySemy(i).Value
                IF PolyValue.Value < .4 THEN PolyValue.Value = .4
                PolySemy(i).Value = PolyValue.Value

                AvgPoly! = AvgPoly! + PolyValue.Value

        ELSE

                PolyValue.Value = 0

        END IF

        PUT #1, i, PolyValue

NEXT
AvgPoly! = AvgPoly! / NumRel

CLOSE #1
OPEN NdxDir$ + "PolyAvg.dat" FOR OUTPUT AS #1
        PRINT #1, AvgPoly!
CLOSE #1

'=============================================================

IF INKEY$ = CHR$(27) GOTO PolySemyDone

SortT PolySemy(1), NumRel, DESCEND, LEN(PolySemy(1)), 2, -3

OPEN LstDir$ + "PolySemy.LST" FOR OUTPUT AS 1
PRINT #1, "New Value Freq  #RelList TotRelVal Rel/Freq OrigPoly Code  Word/Phrase"
'         "###.##### #####    #####     #####  ###.### ####.### ##### &"

'PRINT #1, "PolyValue Word/Phrase"
'         "###.##### &"

FOR i = 1 TO NumRel

        CALL QPrintRC(STR$(i), 12, 32, -1)

        PolyFreq = Ndx(PolySemy(i).Code).Num

        IF PolySemy(i).Value > 0 AND PolyFreq > 3 THEN

              '** FGetRT RelInvNdxFile, RelInvNdx, CLNG(PolySemy(i).Code), LEN(RelInvNdx)

              'PRINT #1, USING "###.##### &"; PolySemy(i).Value; Dict$(PolySemy(i).Code)
              TotRelVal& = TotRelValue(PolySemy(i).Code)
              IF TotRelVal& < 0 THEN TotRelVal& = TotRelVal& + 65536
              PRINT #1, USING "###.##### #####    #####     #####  ###.### ####.### ##### &"; PolySemy(i).Value; PolyFreq; RelInvNdx
Code; Dict$(PolySemy(i).Code)

        END IF

NEXT

CLOSE 1

PolySemyDone:
```

```
ELSE
        ULr = 9
END IF
        DULr = ULr - 1
        LRr = ULr + Lin + 1
        DLRr = LRr + 2

horizmargin = (80 - Wid) \ 2
        ULc = horizmargin
        DULc = ULc - 3
        LRc = 80 - horizmargin
        IF (Wid \ 2) * 2 = Wid THEN LRc = LRc + 1' if it's even WIDth, then bump up the LRcolumn
        DLRc = LRc + 1

REDIM Scr%(ArraySize%(DULr, DULc, DLRr, DLRc))
CALL ScrnSave0(DULr, DULc, DLRr, DLRc, SEG Scr%(0))
CALL WindMgr(ULr, ULc, LRr, LRc, 4, NormAttr, RevAttr, "Status")
FOR i = 1 TO Lin
        CALL QPrintRC(Text$(i), ULr + i, ULc + 1, -1)
NEXT

r = ULr + Lin
c = ULc + 1 + LEN(Text$(Lin))
IF LEN(Text$(Lin)) + 2 = Wid THEN c = ULc + 1: r = r + 1

ERASE Text$
WindOpen = TRUE

EXIT SUB
'------------------------------------------------------------ close window
MsgClose:

CALL ScrnRest0(DULr, DULc, DLRr, DLRc, SEG Scr%(0))
ERASE Scr%
WindOpen = FALSE
RETURN

END SUB

SUB EmsAlloc (NumPgs%, HandleX, FileName$) STATIC

IF EmsPagesFree% < NumPgs THEN
        CALL Chime(8)
        PRINT "Couldn't Allocate "; CLNG(NumPgs) * SixteenK; " bytes of EMS ("; NumPgs; " pages) for "; FileName$
        PRINT "Only "; EmsPagesFree%; " pages were available."
        CALL ReleaseEMS
        STOP
ELSE
        CALL EmsAllocMem(NumPgs%, HandleX)
END IF

END SUB

FUNCTION LoadIntoEMS (File$) STATIC

'-------- Returns the EMS handle where the file was loaded into --------

EMSPg = EmsGetPFSeg%
SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, FileEMS, File$

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
FOpenAll File$, 0, 4, LoadFILE
Box0 14, 8, 18, 72, 2, Re.Attr
PaintBox0 14, 8, 18, 72, RevAttr
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC "Loading " + File$ + " block" + STR$(i) + "/" + STR$(Num32kBlocks - 1) + " ", 16, 10, RevAttr

        '-- map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapPage FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT

        '-- seek to beginning of current block
        FSeek LoadFILE, (i - 1) * ThirtyTwoK&

        IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        IF i < Num32kBlocks + 1 THEN

                '-- get the 32k block and put it directly into the EMS page frame
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        ELSE

                '-- load the left over (<32k) bytes
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, LeftOver&

                IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

        END IF
NEXT
```

```
FClose LoadFILE

ClearScr0 14, 8, 18, 72, NormAttr

LoadIntoEMS = FileEMS

END FUNCTION

SUB ReleaseEMS STATIC

IF DictWordEMS THEN CALL EmsRelMem(DictWordEMS)
IF RelInvDatEMS THEN CALL EmsRelMem(RelInvDatEMS)

END SUB

SUB WindMgr (ULRow, ULCol, LRRow, LRCol, Frame, BoxColr, TextColr, Text$) STATIC

CALL Box0(ULRow - 1, ULCol - 1, LRRow + 1, LRCol + 1, Frame, BoxColr)
CALL ClearScr0(ULRow, ULCol, LRRow, LRCol, BoxColr)
CALL QPrintRC("[" + Text$ + "]", ULRow - 1, ULCol + 1, TextColr)

END SUB

DEFINT A-Z

' ============
'  NEWKEY.BAS
' ============

'Invoked: NEWKEY ConfigFile

'Creates: NewKey.Ndx, NewVal.Ndx

'Uses: Key.Ndx, Weight.Ndx, PolySemy.Dat, KyInvrts.Ndx

'-- NewVal.Ndx contains new weights for Doc n sorted by a new weight
'   where NewWeight = Weight * (Poly ^ .125)

'  TYPE WeightAvgNdx127
'     Weight(1 TO 127) AS SINGLE
'     Mult AS SINGLE
'  Mult is used to vary the number of sentences in the abstract program

'-- NewKey.Ndx contains the corresponding codes for Doc n
'   i.e., sorted by their new weights

'  TYPE KeyNdx127
'     Num AS INTEGER
'     Code(1 TO 127) AS INTEGER

'$INCLUDE: '\\vadim\c-drive\USER\INCLUDE\TYPES.BI'
'$INCLUDE: '\\vadim\c-drive\USER\INCLUDE\DECLARES.BI'

TYPE ValCodeType
        Value AS SINGLE
        Code AS INTEGER
END TYPE

'TYPE MultType
'  Code AS INTEGER
'  Num AS INTEGER
'END TYPE

CONST FALSE = 0, TRUE = NOT FALSE

DECLARE FUNCTION LoadIntoEMS% (File$, NoEMSFlag%)
DECLARE SUB NewKey ()
DECLARE SUB Config ()
DECLARE SUB EmsAlloc (NumPages%, Handle%, LoadFILE$)
DECLARE SUB LoadData ()
DECLARE SUB ReleaseEMS ()


'---------------- COMMON SHARED Variables/Arrays ----------------

' Color Attributes
COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr

' Constants
COMMON SHARED ThirtyTwoKB, SixtyFour, SixteenK, sing, Comb, BlankBit$

' Directories
COMMON SHARED NdxDir$, DocDir$, KeyDir$, LstDir$, AbstrDir$

' Data file sizes and EMS handles

COMMON SHARED KeyNdxFILE
COMMON SHARED PolysemyEMS, PolyValue AS PolyValueType, PolyLen
COMMON SHARED KeyEMS, WeightEMS, PolyEMS
COMMON SHARED KeyFile, WeightFile, PolyFile
COMMON SHARED WeightTemp AS WeightNdx127, WeightLen
COMMON SHARED KeyTemp AS KeyNdx127, KeyLen
COMMON SHARED ValArray AS WeightAvgNdx127, ValLen
COMMON SHARED LenTemp, NewKeyFile, NewValueFile, NewValueEMS
COMMON SHARED PolyNoEMSFlag, KeyNoEMSFlag, WeightNoEMSFlag
'---------- MAIN MODULE ----------

CALL Config
ValLen = LEN(ValArray)
CALL LoadData
```

```
File$ = NdxDir$ + "NewKey.ndx"
FOpenAll File$, 1, 4, NewKeyFile
'-- if the file wasn't there, create it
IF NewKeyFile = -1 THEN
        FCreate File$
        FOpenAll File$, 1, 4, NewKeyFile
END IF

File$ = NdxDir$ + "NewVal.ndx"
FOpenAll File$, 1, 4, NewValueFile
'-- if the file wasn't there, create it
IF NewValueFile = -1 THEN
        FCreate File$
        FOpenAll File$, 1, 4, NewValueFile
END IF

QPrintRC "NEWKEY IS WORKING", 2, 30, -1

CALL NewKey
CALL ReleaseEMS

END

SUB Config STATIC

        IF COMMAND$ <> "" THEN
                ConfigFile$ = COMMAND$
                SpcLoc = INSTR(ConfigFile$, " ")
                IF SpcLoc THEN ConfigFile$ = LEFT$(ConfigFile$, SpcLoc - 1)
                ConfigFile$ = ConfigFile$ + ".CFG"
        ELSE
                Chime 5
                PRINT "Usage NEWKEY [database]."
                STOP
        END IF

        IF NOT Exist(ConfigFile$) THEN
                PRINT ConfigFile$; " was not found."
                PRINT "Press any key to continue."
                DO: LOOP UNTIL LEN(INKEY$)
                END
        END IF
        OPEN ConfigFile$ FOR INPUT AS #1
                        INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
        CLOSE #1

NormAttr = OneColor%(Fg, Bg)
RevAttr = OneColor%(Bg, Fg AND 7)
SixteenK = 16384
ThirtyTwoK& = SixteenK * 2
CLS

IF NOT EmsLoaded% THEN
        Chime 8
        PRINT "The EMS has not been loaded."
        STOP
END IF
END SUB

SUB EmsAlloc (NumPages%, Handle%, LoadFILE$) STATIC

CALL EmsAllocMem(NumPages%, Handle%)
IF EmsError% THEN
        PRINT "Couldn't allocate"; CLNG(NumPages) * SixteenK; "bytes of EMS for "; LoadFILE$
        PRINT "Use disk space."
        Handle = 0

END IF

END SUB
```

```
SUB LoadData STATIC

'---- Load Weight.Ndx into EMS
File$ = NdxDir$ + "WEIGHT.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
WeightLen = LEN(WeightTemp)
WeightEMS = LoadIntoEMS(File$, WeightNoEMSFlag)
IF WeightNoEMSFlag THEN  'use disk
        FOpenAll File$, 0, 4, WeightFile
END IF

'---- Load Key.Ndx into EMS
File$ = NdxDir$ + "KEY.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
KeyLen = LEN(KeyTemp)
KeyEMS = LoadIntoEMS(File$, KeyNoEMSFlag)
IF KeyNoEMSFlag THEN
                FOpenAll File$, 0, 4, KeyFile
END IF
```

```
'—— Load PolySemy.DAT into EMS
File$ = NdxDir$ + "POLYSEMY.DAT"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
DIM PolyVal AS PolyValueType
PolyLen = LEN(PolyVal)
PolyEMS = LoadIntoEMS(File$, PolyNoEMSFlag)
IF PolyNoEMSFlag THEN
                FOpenAll File$, 0, 4, PolyFile
END IF

'NumDoc& = FileSize&(NdxDir$ + "KEY.NDX") / KeyLen
'NumPages = NumDoc& * CLNG(ValLen) \ SixteenK + 2' round off to nearest 2 pages
'EmsAlloc NumPages, NewValueEMS, "NewVal.Ndx"
'FOpenAll File$, 0, 4, NewValueEMS
END SUB

FUNCTION LoadIntoEMS (File$, NoEMSFlag) STATIC

'———— Returns the handle where the file was loaded into ————
NoEMSFlag = FALSE
EMSPg = EmsGetPFSeg%
SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, FileEMS, File$
IF FileEMS = 0 THEN 'not enough EMS use disk instead
        NoEMSFlag = TRUE
        EXIT FUNCTION
END IF
Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
FOpenAll File$, 0, 4, LoadFILE
FOR i = 1 TO Num32kBlocks + 1
        Box0 11, 5, 15, 70, 2, RevAttr
        'PaintBox0 11, 5, 15, 70, RevAttr
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1), 13, 12, RevAttr

        '— map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
        NEXT
        '— seek to beginning of current block
        FSeek LoadFILE, (i - 1) * ThirtyTwoK&
        IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP
        IF i < Num32kBlocks + 1 THEN
                '— get the 32k block and put it directly into the EMS page frame
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&

                IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        ELSE
                '— load the left over (<32k) bytes
                FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, LeftOver&
                IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        END IF
NEXT

FClose LoadFILE

CLS

LoadIntoEMS = FileEMS

END FUNCTION

SUB NewKey STATIC

REDIM AvgRatio!(1 TO 127)

AvgV30! = 0

DocNum& = FileSize&(NdxDir$ + "Key.ndx") \ LEN(KeyTemp)
FOR RecNum& = 1 TO DocNum&

        QPrintRC "Document #" + STR$(RecNum&), 10, 30, -1
        IF KeyNoEMSFlag THEN
                FGetRT KeyFile, KeyTemp, RecNum&, KeyLen
        ELSE
                EmsGet KeyTemp, KeyLen, RecNum&, KeyEMS
        END IF

        IF WeightNoEMSFlag THEN
                FGetRT WeightFile, WeightTemp, RecNum&, WeightLen
        ELSE
                EmsGet WeightTemp, WeightLen, RecNum&, WeightEMS
        END IF

        IF KeyTemp.Num = 0 THEN KeyTemp.Num = 127

        REDIM TempArray(1 TO KeyTemp.Num) AS ValCodeType

        FOR i = 1 TO KeyTemp.Num
                IF PolyNoEMSFlag THEN
                        FGetRT PolyFile, PolyValue, CLNG(KeyTemp.Code(i)), PolyLen
```

```
                ELSE
                        EmsGet PolyValue, PolyLen, CLNG(KeyTemp.Code(i)), PolyEMS
                END IF
                TempArray(i).Value = WeightTemp.Weight(i) * PolyValue.Value * .125  '(i.e. *1/8)
                TempArray(i).Code = KeyTemp.Code(i)
        NEXT

        SortT TempArray(1), KeyTemp.Num, 1, LEN(TempArray(1)), 0, -3

        KeyTemp.Num = KeyTemp.Num
        FOR k = 1 TO KeyTemp.Num
                KeyTemp.Code(k) = TempArray(k).Code
                ValArray.Weight(k) = TempArray(k).Value
        NEXT

        FOR f = k TO 127
                KeyTemp.Code(f) = 0
                ValArray.Weight(f) = 0
        NEXT

        ValArray.Mult = 1

        FPutRT NewValueFile, ValArray, RecNum&, ValLen
        FPutRT NewKeyFile, KeyTemp, RecNum&, KeyLen

        IF INKEY$ = CHR$(27) THEN EXIT SUB

NEXT 'document

CLS
PRINT "Everything is OK"
CALL ReleaseEMS
FClose NewKeyFile
FClose NewValueFile

END SUB

SUB ReleaseEMS STATIC

IF KeyEMS THEN EmsRelMem KeyEMS
IF WeightEMS THEN EmsRelMem WeightEMS
IF PolyEMS THEN EmsRelMem PolyEMS
IF KeyFile THEN FClose KeyEMS
IF WeightFile THEN FClose WeightEMS
IF PolyFile THEN FClose PolyFile

END SUB

DEFINT A-Z
'CHANGEFL.BAS
'Invoked changefl ConfigFile
'Creates: KyInvrts.Ndx, Rel-Invs.Ndx
'  TYPE SmallNdxType
'      Index AS LONG
'      Num AS INTEGER
'Uses: KyInvert.Ndx, RelInv.Ndx
'  TYPE NdxType
'      Code AS INTEGER
'      Index AS LONG
'      Num AS INTEGER
'makes copies of KyInvert.Ndx & Rel-Inv.Ndx without the Code field

'$INCLUDE: '\USER\INCLUDE\TYPES.BI'
'$INCLUDE: '\USER\INCLUDE\dECLARES.BI'
        IF COMMAND$ <> "" THEN
                ConfigFile$ = COMMAND$
                SpcLoc = INSTR(ConfigFile$, " ")
                IF SpcLoc THEN ConfigFile$ = LEFT$(ConfigFile$, SpcLoc - 1)
                ConfigFile$ = ConfigFile$ + ".CFG"
        ELSE
                Chime 5
                PRINT "Usage CHANGEFL [dataname]."
                END
        END IF

        OPEN ConfigFile$ FOR INPUT AS #1
                INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
        CLOSE #1
QPrintRC "CHANGEFL", 1, 37, -1
file$ = NdxDir$ + "REL-INV.NDX"
IF NOT ExistX(file$) THEN
        CLS
        PRINT file$; " was not found."
        END
END IF
NdxLEN = 8
FreqCompNum = FileSize&(file$) \ NdxLEN
REDIM TempFCIndx(1 TO FreqCompNum) AS NdxType
DIM FCIndx AS SmallNdxType
FGetAH file$, TempFCIndx(1), NdxLEN, FreqCompNum
CLS
OPEN NdxDir$ + "Rel-InvS.NDX" FOR RANDOM AS #1 LEN = 6

FOR i = 1 TO FreqCompNum
        FCIndx.Index = TempFCIndx(i).Index
        FCIndx.Num = TempFCIndx(i).Num
        LOCATE 10, 25
        PRINT i
        PUT #1, i, FCIndx
NEXT
ERASE TempFCIndx
CLOSE #1
```

```
' Load the Inverted Data Files

file$ = NdxDir$ + "KYINVERT.NDX"
IF NOT ExistX(file$) THEN
        CLS
        PRINT file$; " was not found."
        END
END IF
NumKeys = FileSize&(file$) \ NdxLEN
REDIM TempKYIndx(1 TO NumKeys) AS NdxType
FGetAH file$, SEG TempKYIndx(1), NdxLEN, NumKeys
DIM KYIndx  AS SmallNdxType
OPEN NdxDir$ + "KYINVRTS.NDX" FOR RANDOM AS #1 LEN = 6
FOR i = 1 TO NumKeys
        KYIndx.Index = TempKYIndx(i).Index
        KYIndx.Num = TempKYIndx(i).Num
        LOCATE 10, 25
        PRINT i
        PUT #1, i, KYIndx
NEXT
ERASE TempKYIndx
CLS
CLOSE #1
END


DEFINT A-Z
'$INCLUDE: '\USER\INCLUDE\TYPES.BI'
'$INCLUDE: '\USER\INCLUDE\DECLARES.BI'

CONST MaxText = 2500, ASC0 = 48, ASC9 = 57, FALSE = 0, TRUE = NOT FALSE
CONST SixteenK = 16384, ThirtyTwoK& = 32768

TYPE WrdLen
        Wrd AS INTEGER
        Len AS INTEGER
END TYPE

DECLARE FUNCTION GetSentValue! (SentNum, NumWord, Value AS WeightAvgNdx127, KeyTemp AS KeyNdx127)
DECLARE FUNCTION InstrTbl% (Start%, Source$, Char$)
DECLARE FUNCTION LoadIntoEMS% (File$)
DECLARE FUNCTION ExtractWord$ (Source$, Char$, Start%)
DECLARE SUB ChangeChar (Txt$, KeepStr$)
DECLARE SUB Config (Machine$, Beg&, Fin&)
DECLARE SUB Chopping (Best)
DECLARE SUB CutSentence (BestSent, Sent$, NumWord)
DECLARE SUB DictSortSearch (KeyTemp AS KeyNdx127, KeepNum, SingList() AS WordCode, CombList() AS WordCode)
DECLARE SUB DosErrHandler ()
DECLARE SUB ExtractDoc (DocNum&, TotSentNum%, HandleX, NoTextFlag)
DECLARE SUB ExtractFullText (TxtEMS%, FullEMS%)
DECLARE SUB ExtractSent (SentNum%, Sent$, LenSen%, HandleX)
DECLARE SUB ExtractWordNum (Source$, WordNum%, Start%, Slen%)
DECLARE SUB EmsAlloc (NumPages%, HandleX, LoadFILE$)
DECLARE SUB FillScrn0 (UL, LC, BL, RC, Colr, Char)
DECLARE SUB FindCombKey (WordList$(), NumWord%, NumKeyWordFound%, SentNum, NumKey, CombList() AS WordCode)
DECLARE SUB FindSingKey (WordList$(), NumWord%, NumKeyWordFound%, SentNum, NumKey, SingList() AS WordCode)
DECLARE SUB GetKWList (DocNum&, SingList() AS WordCode, CombList() AS WordCode, KeyText AS KeyNdx127)
DECLARE SUB LoadData ()
DECLARE SUB Rank (BestSent, NoMoreFlag, NumKeyWords(), NumWord(), Value AS WeightAvgNdx127, KeyTemp AS KeyNdx127)
DECLARE SUB ReadEnglishText (FirstLine&, LastLine&, HandleX, TotSentNum%)
DECLARE SUB ReadGermanText (FirstLine&, LastLine&, HandleX, TotSentNum%)
DECLARE SUB ReleaseEMS ()
DECLARE SUB ReadSection (Txt$, SecArray$(), ArtArray$())
DECLARE SUB WordParse (Sent$, LenSen, SentNum%, WordList$(), Words%)
DECLARE SUB WriteSentence (Sent$, Best, AbstrPos&)

'---------------- COMMON SHARED Variables/Arrays----------------

COMMON SHARED Fg, Bg, Brdr, NormAttr, RevAttr
COMMON SHARED LstDir$, NdxDir$, DocDir$, AbstrDir$, AtList$, Lang$
COMMON SHARED DictCodeNum%, DictCodeHI%, KeyEMS, ValueEMS, NumComb, NumSing
COMMON SHARED DocNdxFile, DocFile, AbstrNdxFile, AbstrFile, TextNum&
COMMON SHARED KeyWordFoundNdx() AS WordNdxType, KeyWordFound() AS ListType
COMMON SHARED SentNdx() AS SentNdxType, TextArray() AS TextType, TotSentNum
COMMON SHARED KeepNoNumbers$, Keep$, Abbrev1$(), Noise$(), AbbrevEngl$()
COMMON SHARED Section$(), Article$(), Paragraph$(), Artikel$(), Numbers$()
COMMON SHARED MeanPrefixes$(), Prefixes$()
'----------MAIN MODULE----------

CALL Config(Machine$, Beg&, Fin&)

CALL LoadData
File$ = DocDir$ + ".NDX"
FOpenAll File$, 0, 2, DocNdxFile
File$ = DocDir$ + ".TXT"
FOpenAll File$, 0, 2, DocFile
File$ = AbstrDir$ + ".ND" + Machine$
FCreate File$
FOpenAll File$, 1, 4, AbstrNdxFile
File$ = AbstrDir$ + ".TX" + Machine$
FCreate File$
FOpenAll File$, 1, 4, AbstrFile

CountAll& = FileSize&(DocDir$ + ".NDX") \ 8' number of records (i.e., files)

'ON ERROR GOTO ErrorHandler

DIM Abstr AS ISAMtype
LenAbstr = LEN(Abstr)
AbstrPos& = 1
a11# = TIMER
```

```
FOR DocNum& = Beg& TO Fin& 'CountAll&   '— IN REAL PROGRAMM TO CountAll&
        PRINT FRE("")
        NumKey = 0
        CALL ExtractDoc(DocNum&, TotSentNum, TxtEMS%, NoTextFlag)
        IF NoTextFlag THEN NoTextFlag = FALSE: GOTO NextDoc'if no text in ndx file for such doc. num.(doc. 3291 in west)
        REDIM NumWord(1 TO TotSentNum)
        REDIM NumKeyWords(1 TO TotSentNum)
        REDIM KeyWordFoundNdx(1 TO TotSentNum) AS WordNdxType
        REDIM KeyWordFound(1 TO 1) AS ListType
        REDIM SingList(1 TO 1) AS WordCode
        REDIM CombList(1 TO 1) AS WordCode
        DIM KeyTemp AS KeyNdx127
        CALL GetKWList(DocNum&, SingList(), CombList(), KeyTemp)
        UL = 6: LC = 25: BL = 9: RC = 55
        Box0 UL, LC, BL, RC, 2, RevAttr
        PaintBox0 UL, LC, BL, RC, RevAttr
        QPrintRC "Document  " + STR$(DocNum&), UL + 1, LC + 9, RevAttr
        QPrintRC "Total Sentences" + STR$(TotSentNum), UL + 2, LC + 5, RevAttr
        UL = 12: LC = 30: BL = 17: RC = 50
        Box0 UL, LC, BL, RC, 2, RevAttr
        FOR SentNum = 1 TO TotSentNum  'How many sentences?
                CALL ExtractSent(SentNum, Sent$, LenSen, TxtEMS%)
                REDIM WordList$(1 TO 1)
                CALL WordParse(Sent$, LenSen, SentNum, WordList$(), NumWord(SentNum))
                IF NumWord(SentNum) > 5 THEN
                        CALL FindCombKey(WordList$(), NumWord(SentNum), NumKeyWords(SentNum), SentNum, NumKey, CombList())
                        CALL FindSingKey(WordList$(), NumWord(SentNum), NumKeyWords(SentNum), SentNum, NumKey, SingList())
                        FillScrn0 UL + 1, LC + 1, BL - 1, RC - 1, RevAttr, 32
                        QPrintRC "Sentence " + STR$(SentNum), UL + 1, LC + 5, RevAttr
                        QPrintRC "Words    " + STR$(NumWord(SentNum)), UL + 2, LC + 5, RevAttr
                        QPrintRC "KeyWords " + STR$(NumKeyWords(SentNum)), UL + 3, LC + 5, RevAttr
                        ERASE WordList$
                END IF
        NEXT
        'erase text without punctuation and extract the full text
        CALL ExtractFullText(TxtEMS, FullEMS)
'--------create abstract and write it onto the disk
        CLS
        UL = 4: LC = 25: BL = 7: RC = 55
        Box0 UL, LC, BL, RC, 2, RevAttr
        PaintBox0 UL, LC, BL, RC, RevAttr
        QPrintRC "Document  " + STR$(DocNum&), UL + 1, LC + 9, RevAttr
        QPrintRC "Total Sentences" + STR$(TotSentNum), UL + 2, LC + 5, RevAttr
        LENTxt = 80
        Abstr.First = AbstrPos&
        NoMoreFlag = FALSE
        DIM WordValue AS WeightAvgNdx127
'   EMSGet WordValue, LEN(WordValue), DocNum&, ValueEMS
        FGetRT ValueEMS, WordValue, DocNum&, LEN(WordValue)
'   OPEN "Origin" FOR OUTPUT AS #8
'   OPEN "New" FOR OUTPUT AS #9
        DO
                REDIM KeyWords$(1 TO 1)
                CALL Rank(BestSent, NoMoreFlag, NumKeyWords(), NumWord(), WordValue, KeyTemp)
                IF NoMoreFlag THEN EXIT DO
                CALL ExtractSent(BestSent, Sent$, LenSen, FullEMS%)
'     PRINT #8, Sent$
                CALL CutSentence(BestSent, Sent$, NumWord)
                CALL WriteSentence(Sent$, BestSent, AbstrPos&)
                CALL Chopping(BestSent)
        LOOP UNTIL NoMoreFlag
'   CLOSE #8, #9
        Abstr.Last = AbstrPos& - 1
        FPutRT AbstrNdxFile, Abstr, DocNum&, 8
        CALL DosErrHandler
        IF TextNum& > MaxText THEN
                CALL EmsRelMem(FullEMS%)
        ELSE
                ERASE TextArray
                FreeSpace$ = FRE("")
        END IF
        cht$ = INKEY$
        IF LEFT$(cht$, 1) = CHR$(27) THEN
                GOTO FinishWork
        END IF
NextDoc:
ERASE NumWord, NumKeyWords, KeyWordFound, KeyWordFoundNdx, SingList, CombList
CALL FFlush(AbstrNdxFile)
CALL FFlush(AbstrFile)
CALL FFlush(DocNdxFile)
CALL FFlush(DocNdxFile)
NEXT 'document
FinishWork:
a12# = TIMER - a11#
H = a12# \ 3600
M = (a12# MOD 3600) \ 60
s = ((a12# MOD 3600) MOD 60)
QPrintRC "Total  : " + STR$(H) + " H " + STR$(M) + " m " + STR$(s) + " s", 19, 25, NormAttr
QPrintRC "Total  : " + STR$(a12#), 20, 25, NormAttr
FClose AbstrNdxFile
FClose AbstrFile
FClose DocNdxFile
FClose DocFile
ReleaseEMS

END 'program

'-------------------END OF MAIN MODULE-------------------------------
```

```
'------------------ DATA
GermanAbbreviature:
DATA "zB","ggt","inkl.","gl","grunds","ausschl","einschl"
DATA "Kl","Bekl","Nr","Ge:","BerGer","ff","subj","obj"
EnglAbbreviature:
DATA "mr","mrs","messrs","sen","rep","ms","dr","drs"
EnglSections:
DATA "Section","Sec","Sec."
DATA "Article","Art",Art."
GermanSections:
DATA "Paragraph","Par","Par.","Para."
DATA "Artikel","Art","Art."
NumData:
DATA "I","II","III","IV","V","VI","VII","VIII","IX","X"
DATA "XI","XII","XIII","XIV","XV","XVI","XVII","XVIII","XIX","XX"
DATA "XXI","XXII","XXIII","XXIV","XXV","XXVI","XXVII","XXVIII","XXIX","XXX"

'$INCLUDE: '\user\include\prefixes.bi'

SUB ChangeChar (Txt$, KeepStr$) STATIC

'-- replace all chars except contained in Keep$
New$ = " "
FOR i = 1 TO LEN(Txt$)
        IF INSTR(KeepStr$, MID$(Txt$, i, 1)) = 0 THEN
                MID$(Txt$, i, 1) = New$
        END IF
NEXT

END SUB

SUB Chopping (Best) STATIC
'------delete kw which is found already from the kwlist
        NumCodes = KeyWordFoundNdx(Best).Last - KeyWordFoundNdx(Best).First + 1
        REDIM CodeDelete(1 TO NumCodes) 'array of the codes which should be deleted
        j = 0
        FOR i = KeyWordFoundNdx(Best).First TO KeyWordFoundNdx(Best).Last
             IF KeyWordFound(i).Code <> 0 THEN
                        j = j + 1
                        CodeDelete(j) = KeyWordFound(i).Code
                ELSE
                        NumCodes = NumCodes - 1
                END IF
        NEXT
        FOR i = 1 TO UBOUND(KeyWordFound)
                CALL Search(SEG CodeDelete(1), NumCodes, KeyWordFound(i).Code, Found, 0, 0, -1)
                IF Found <> -1 THEN 'delete this word
'          KeyWordFound(i).Code = 0
                        KeyWordFound(i).Num = 0
                    END IF
        NEXT
END SUB

SUB Config (Machine$, Beg&, Fin&) STATIC
        Cmd$ = RTRIM$(LTRIM$(COMMAND$))
        Parms = InCount(Cmd$, " ") + 1   '-- number of parameters

IF Parms = 4 THEN

       '-- Expected information on command line:
       '   Config file, First Doc, Last Doc

       Extract Cmd$, " ", 1, Strt, Slen '-- extract first parm
       DBName$ = MID$(Cmd$, Strt, Slen)
       ConfigFile$ = DBName$ + ".CFG"

       Extract Cmd$, " ", 2, Strt, Slen '-- extract second parm
       Machine$ = MID$(Cmd$, Strt, Slen)

       Extract Cmd$, " ", 3, Strt, Slen '-- extract third parm
       Beg& = QPVal&(MID$(Cmd$, Strt, Slen))

       Extract Cmd$, " ", 4, Strt, Slen '-- extract fourth parm
       Fin& = QPVal&(MID$(Cmd$, Strt, Slen))

ELSE
       PRINT
       PRINT "ABSTRACT Program Error: Missing Parameters"
       PRINT
       PRINT
       PRINT "Required Parameters are:"
       PRINT
       PRINT "Abstr  Config File  First Doc  Last Doc"
       PRINT
       Chime 10
       PRINT "Press the SPACE BAR to exit:"
       i$ = INPUT$(1)
       END
END IF

OPEN ConfigFile$ FOR INPUT ACCESS READ SHARED AS #1
  INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
CLOSE #1

OPEN LstDir$ + "a" + Lang$ + ".LST" FOR INPUT AS #1
       LINE INPUT #1, At_list$
CLOSE #1

NormAttr = OneColor%(Fg, Bg)
RevAttr = OneColor%(Bg, Fg AND 7)
CLS
```

```
' replacement string for punctuation
Keep$ = " "
KeepNoNumbers$ = " "
FOR j = 65 TO 90: Keep$ = Keep$ + CHR$(j): NEXT
FOR j = 97 TO 122: Keep$ = Keep$ + CHR$(j): NEXT
Keep$ = Keep$ + "'ß"
KeepNoNumbers$ = Keep$
FOR j = 48 TO 57: Keep$ = Keep$ + CHR$(j): NEXT
RESTORE GermanAbbreviature
REDIM Abbrev1$(1 TO 15)
FOR i = 1 TO 15
  READ Abbrev1$(i)
NEXT
i = 1
REDIM Noise$(1 TO 1)
IF Lang$ = "GERMAN" THEN
        OPEN LstDir$ + "NOISE.DAT" FOR INPUT ACCESS READ SHARED AS #1
        DO UNTIL EOF(1)
          REDIM PRESERVE Noise$(1 TO i)
          INPUT #1, Noise$(i)
          n$ = LEFT$(Noise$(i), LEN(Noise$(i)) - 1)
          Noise$(i) = UCASE$(LEFT$(n$, 1)) + MID$(n$, 2)
          i = i + 1
        LOOP
        CLOSE #1
END IF
IF NOT EmsLoaded% THEN
        Chime 8
        PRINT "The EMS Driver has not been Loaded."
        STOP
END IF

DIM AbbrevEngl$(1 TO 8)
FOR i = 1 TO 8
        READ AbbrevEngl$(i)
NEXT
DIM Section$(1 TO 3)
FOR i = 1 TO 3
        READ Section$(i)
NEXT
DIM Article$(1 TO 3)
FOR i = 1 TO 3
        READ Article$(i)
NEXT
DIM Paragraph$(1 TO 4)
FOR i = 1 TO 4
        READ Paragraph$(i)
NEXT
DIM Artikel$(1 TO 3)
FOR i = 1 TO 3
        READ Artikel$(i)
NEXT
DIM Numbers$(1 TO 30)
FOR i = 1 TO 30
        READ Numbers$(i)
NEXT

IF Lang$ = "GERMAN" THEN
        RESTORE GermanPrefixes
ELSE
        RESTORE EnglishPrefixes
END IF
REDIM Prefixes$(2 TO 9)
IF Lang$ = "GERMAN" THEN
        FOR i = 2 TO 9
                READ FirstHalf$, SecondHalf$
                Prefixes$(i) = FirstHalf$ + SecondHalf$
        NEXT
ELSE
        FOR i = 2 TO 9
                READ Prefixes$(i)
        NEXT
END IF

REDIM MeanPrefixes$(3 TO 14)
IF Lang$ = "GERMAN" THEN
        FOR i = 3 TO 14
                READ FirstHalf$, SecondHalf$, ThirdHalf$
                MeanPrefixes$(i) = FirstHalf$ + SecondHalf$ + ThirdHalf$
        NEXT
ELSE
        FOR i = 3 TO 14
                READ MeanPrefixes$(i)
        NEXT
END IF

END SUB

SUB CutSentence (BestSent, Sent$, NumWord) STATIC

'this procedure leaves in the sentence only keywords with 30 characters of
'both sides but always full word and changes the rest with "..."
' 1. Collect the keywords into array
' 2. Find this kw's in the sentence
' 3. Take 40 characters (but full word) of both sides and if there is
'    something else then cut it

        n = 0: OldStr$ = ""
        'save numbers into array
```

```
NumCodes = KeyWordFoundNdx(BestSent).Last - KeyWordFoundNdx(BestSent).First + 1
REDIM KeyWords$(1 TO NumCodes)
FOR i = KeyWordFoundNdx(BestSent).First TO KeyWordFoundNdx(BestSent).Last
        IF KeyWordFound(i).Num <> 0 THEN
                IF INSTR(OldStr$, RTRIM$(KeyWordFound(i).Str) + " ") = 0 THEN
                        n = n + 1
                        KeyWords$(n) = RTRIM$(KeyWordFound(i).Str)
                        OldStr$ = OldStr$ + KeyWords$(n) + " "
                END IF
        END IF
NEXT
REDIM PRESERVE KeyWords$(1 TO n)

'for each kw find position
SentClean$ = Sent$
CALL ChangeChar(SentClean$, KeepNoNumbers$)
IF Lang$ = "GERMAN" THEN CALL Lower(SentClean$)
NumKey = 0
REDIM KeyStart(1 TO NumCodes) AS WrdLen
FirstTime = TRUE
FOR i = 1 TO n
        Start = 1
        DO
                NumKey = NumKey + 1
                'check the first word with capital letter before
                IF Lang$ = "GERMAN" THEN
                        KeyStart(NumKey).Wrd = INSTR(Start, SentClean$, KeyWords$(i) + " ")
                ELSE
                        IF FirstTime THEN
                                KeyStart(NumKey).Wrd = INSTR(SentClean$, RTRIM$(UCASE$(LEFT$(KeyWords$(i), 1)) + MID$(KeyWord
                        END IF
                        IF KeyStart(NumKey).Wrd = 0 THEN
                                KeyStart(NumKey).Wrd = INSTR(Start, SentClean$, KeyWords$(i) + " ")
                        END IF
                        'if still not found try add "'s" to the end of the first word
                        IF KeyStart(NumKey).Wrd = 0 THEN
                                SpLoc = INSTR(KeyWords$(i), " ")
                                IF SpLoc THEN
                                        KW$ = LEFT$(KeyWords$(i), SpLoc - 1) + "'s" + MID$(KeyWords$(i), SpLoc)
                                ELSE
                                        KW$ = KeyWords$(i) + "'s"
                                END IF
                                KeyStart(NumKey).Wrd = INSTR(Start, SentClean$, KW$)
                        END IF
                        FirstTime = FALSE
                END IF
                KeyStart(NumKey).Len = LEN(KeyWords$(i))
                IF KeyStart(NumKey).Wrd <> 0 THEN
                        MID$(SentClean$, KeyStart(NumKey).Wrd, KeyStart(NumKey).Len) = STRING$(KeyStart(NumKey).Len, 32)
                        Start = KeyStart(NumKey).Wrd + KeyStart(NumKey).Len
                END IF
        LOOP UNTIL KeyStart(NumKey).Wrd = 0 OR NumKey = NumCodes OR Start >= LEN(Sent$)
        IF KeyStart(NumKey).Wrd = 0 THEN NumKey = NumKey - 1
        IF NumKey = NumCodes THEN EXIT FOR
NEXT

CALL SortT(SEG KeyStart(1), NumKey, 0, LEN(KeyStart(1)), 0, -1)
 j = 0: Shift = 0
 'cut the first part
 IF KeyStart(1).Wrd > 40 THEN
        DelPoint = QInstrB(KeyStart(1).Wrd - 40, Sent$, " ")
        IF DelPoint <> 0 THEN
                Sent$ = "..." + MID$(Sent$, DelPoint)
                Shift = DelPoint - 3
        END IF
 END IF
 'cut the middle parts
 FOR i = 1 TO NumKey - 1
         IF KeyStart(i + 1).Wrd - KeyStart(i).Wrd + KeyStart(i).Len > 80 THEN
                 DelPointStart = INSTR(KeyStart(i).Wrd + KeyStart(i).Len + 40 - Shift, Sent$, " ")
                 DelPointEnd = QInstrB(KeyStart(i + 1).Wrd - Shift - 40, Sent$, " ")
                 IF DelPointStart <> 0 AND DelPointEnd > DelPointStart THEN
                         Sent$ = LEFT$(Sent$, DelPointStart) + "..." + MID$(Sent$, DelPointEnd)
                         Shift = Shift + DelPointEnd - DelPointStart - 3
                 END IF
         END IF
 NEXT
 'cut the last part
 IF KeyStart(NumKey).Wrd <> 0 THEN 'overinsurance
      IF LEN(Sent$) - (KeyStart(NumKey).Wrd + KeyStart(NumKey).Len - Shift) > 40 THEN

                        DelPoint = INSTR(KeyStart(NumKey).Wrd + KeyStart(NumKey).Len - Shift + 40, Sent$, " ")
                        IF DelPoint <> 0 AND DelPoint < LEN(Sent$) THEN
                                Sent$ = LEFT$(Sent$, DelPoint) + "..."
                        END IF
                END IF
        END IF
 END SUB

 SUB DictSortSearch (KeyTemp AS KeyNdx127, KeepNum, SingList() AS WordCode, CombList() AS WordCode) STATIC

 '== Binary searches thru an array of DictType (word & code#) for a
 '== word and returns the code #.  The EMS Handle and the number of
 '== DictCode entries are Common Shared.

 DIM DictTemp AS DictType, DictTemp2 AS DictType
 LENDict = LEN(DictTemp)
 FOR i = 1 TO KeepNum
        Code = KeyTemp.Code(i)
        L = 1: R = DictCodeNum% ' total number of code entries
        DO
```

```
        x = (CLNG(l) + R) \ 2
        EmsGetIEl DictTemp, LENDict, x, DictCodeHX

        IF Code < DictTemp.Code THEN
                R = x - 1
        ELSE
                IF Code <> DictTemp.Code OR Lang$ <> "ENGLISH" THEN
                        l = x + 1
                ELSE
                        '-- save it temporarily in case we need to restore it
                        '   if the forward/backward look doesn't bring any
                        '   positive results (i.e., we didn't get a match, and we
                        '   need the original DictTemp for the LOOP test)
                        SWAP DictTemp2, DictTemp
                        EmsGetIEl DictTemp, LENDict, x + 1, DictCodeHX
                        IF Code = DictTemp.Code THEN
                                x = x + 1
                                EXIT DO
                        ELSE
                                EmsGetIEl DictTemp, LENDict, x - 1, DictCodeHX
                                IF Code = DictTemp.Code THEN
                                        x = x - 1
                                        EXIT DO
                                END IF
                        END IF
                        SWAP DictTemp, DictTemp2
                        l = l + 1
                END IF
        END IF
LOOP UNTIL Code = DictTemp.Code OR l > R
'store previous words with the same code
n = x
DO
  EmsGetIEl DictTemp, LENDict, n, DictCodeHX
  IF DictTemp.Code = Code THEN
                IF INSTR(QPRTrim$(DictTemp.Str), " ") THEN
                        NumComb = NumComb + 1
                        REDIM PRESERVE CombList(1 TO NumComb) AS WordCode
                        CombList(NumComb).Code = DictTemp.Code
                        CombList(NumComb).Str = DictTemp.Str
                ELSE
                        NumSing = NumSing + 1
                        REDIM PRESERVE SingList(1 TO NumSing) AS WordCode
                        SingList(NumSing).Code = DictTemp.Code
                        SingList(NumSing).Str = DictTemp.Str
                END IF
        END IF
        n = n - 1
LOOP UNTIL DictTemp.Code <> Code OR n = 0
n = x + 1
DO
  EmsGetIEl DictTemp, LENDict, n, DictCodeHX
  IF DictTemp.Code = Code THEN
                IF INSTR(QPRTrim$(DictTemp.Str), " ") THEN
                        NumComb = NumComb + 1
                        REDIM PRESERVE CombList(1 TO NumComb) AS WordCode
                        CombList(NumComb).Code = DictTemp.Code
                        CombList(NumComb).Str = DictTemp.Str
                ELSE
                        NumSing = NumSing + 1
                        REDIM PRESERVE SingList(1 TO NumSing) AS WordCode _
                        SingList(NumSing).Code = DictTemp.Code
                        SingList(NumSing).Str = DictTemp.Str
                END IF
                END IF
                n = n + 1
        LOOP UNTIL DictTemp.Code <> Code OR n > DictCodeNum
NEXT
SortT SingList(1), NumSing, 0, 66, 2, 64
SortT CombList(1), NumComb, 0, 66, 2, 64
END SUB

SUB DosErrHandler STATIC
        IF DOSError% THEN
                Chime 5
                PRINT "Error "; ErrorMsg$(WhichError%); " occurred while writing Abstract Index Doc #"; DocNum&
                FClose AbstrNdxFile
                FClose AbstrFile
                FClose DocNdxFile
                FClose DocFile
                ReleaseEMS
                END
        END IF
END SUB

SUB EmsAlloc (NumPages%, HandleX, LoadFILE$) STATIC

CALL EmsAllocMem(NumPages%, HandleX)
IF EmsError% THEN
        PRINT "Couldn't allocate"; CLNG(NumPages) * SixteenK; "bytes of EMS for "; LoadFILE$
        CALL ReleaseEMS
        CALL Chime(2)
        END
END IF

END SUB

SUB ExtractDoc (DocNum&, TotSentNum%, HandleX, NoTextFlag) STATIC
```

```
DIM Doc AS ISAMtype
CLS
Box0 11, 25, 15, 55, 2, RevAttr
PaintBox0 11, 25, 15, 55, RevAttr
QPrintRC "extracting document #" + STR$(DocNum&), 13, 28, RevAttr
Handle = 0
NoTextFlag = FALSE
'---EXTRACT document from file
        FGetRT DocNdxFile, Doc, DocNum&, 8
        TextNum& = Doc.Last& - Doc.First& + 1' lines of text in the file
        IF TextNum& > MaxText THEN
         QPrintRC "*", 2, 2, -1
         '---- Allocate EMS to hold the Text file
         NumPages = 80& * TextNum& / SixteenK + 1' 80 bytes per line,
         CALL EmsAllocMem(NumPages, HandleX)     ' 16K per EMS page
         IF EmsErrorX THEN PRINT "Couldn't allocate"; NumPages * SixteenK; "bytes of EMS.": STOP
         ELSE
                        IF TextNum& <= 0 THEN NoTextFlag = TRUE: EXIT SUB
                        REDIM TextArray(1 TO TextNum&) AS TextType
         END IF
'------------------------- Read Document into EMS eliminating blank lines
'                          change non-alpha chars into spaces
IF Lang$ = "ENGLISH" THEN
        CALL ReadEnglishText(Doc.First&, Doc.Last&, HandleX, TotSentNumX)
ELSEIF Lang$ = "GERMAN" THEN
        CALL ReadGermanText(Doc.First&, Doc.Last&, HandleX, TotSentNumX)
END IF
CLS
END SUB

SUB ExtractFullText (TxtEMS, FullEMS) STATIC
        '---EXTRACT full document from file (should be moved into subroutine)
        DIM Doc AS ISAMtype
        IF TextNum& > MaxText THEN
                CALL EmsRelMem(TxtEMSX)
                '---EXTRACT full document from file
                '---Allocate EMS to hold the Text file
                NumPages = 80& * TextNum& / SixteenK + 1          ' 80 bytes per line,
                CALL EmsAllocMem(NumPages, FullEMSX)              ' 16K per EMS page
                IF EmsErrorX THEN PRINT "Couldn't allocate"; NumPages * SixteenK; "bytes of EMS.": STOP
                NumLines = 0
                DIM Temp1 AS STR80
                LenTemp = LEN(Temp1)
                      FOR i& = Doc.First& TO Doc.Last&
                              FGetRT DocFile, Temp1, i&, 80
                              NumLines = NumLines + 1
                              EmsSet1El Temp1, LenTemp, NumLines, FullEMSX
                      NEXT
        ELSE
                              ERASE TextArray
                              REDIM TextArray(1 TO TextNum&) AS TextType
                              Num = TextNum&
                              Ems2Array TextArray(1), 128, Num, TxtEMS
                              CALL EmsRelMem(TxtEMSX)
        END IF

END SUB

SUB ExtractSent (SentNumX, Sent$, LenSenX, HandleX) STATIC

DIM Temp1  AS STR80
LenTemp = LEN(Temp1)

'----------Get one sentence from the document

  i = SentNdx(SentNum).BL
  IF TextNum& > MaxText THEN
          EmsGet1El Temp1, LenTemp, i, HandleX
  ELSE
          Temp1.Str = TextArray(i).Str
  END IF
  Sent$ = RTRIM$(LTRIM$(RIGHT$(Temp1.Str, 80 - SentNdx(SentNum).BC + 1))) + " "
        IF SentNdx(SentNum).EL > SentNdx(SentNum).BL THEN
                FOR i = SentNdx(SentNum).BL + 1 TO SentNdx(SentNum).EL
                        IF TextNum& > MaxText THEN
                                EmsGet1El Temp1, LenTemp, i, HandleX
                        ELSE
                                Temp1.Str = TextArray(i).Str
                        END IF
                        Sent$ = Sent$ + LTRIM$(RTRIM$((Temp1.Str))) + " "
                NEXT
                l = LEN(RTRIM$(Temp1.Str)) - SentNdx(SentNum).EC
                Sent$ = LEFT$(Sent$, LEN(Sent$) - l)
        ELSE
                Sent$ = LEFT$(Sent$, (SentNdx(SentNum).EC - SentNdx(SentNum).BC + 1))
        END IF
  LenSen = LEN(Sent$)
END SUB

FUNCTION ExtractWord$ (Source$, Char$, Start) STATIC

'extract word from Source$ from Start to Char$

  Source$ = QPTrim$(Source$)
  LenStr = LEN(Source$)
  Slen = 0
        FOR i = Start TO LenStr
                IF MID$(Source$, i, 1) = Char$ AND Slen > 0 THEN
        EXIT FOR
                ELSE
```

```
        Slen = Slen + 1
            END IF
        NEXT
 ExtractWord$ = LTRIM$(MID$(Source$, Start, Slen))
END FUNCTION

SUB ExtractWordNum (Source$, WordNum, Start, Slen) STATIC
SpLoc = 0: Count = 0
DO
  Start = SpLoc + 1
  SpLoc = INSTR(Start, Source$, " ")
  IF SpLoc <> Start THEN
       Count = Count + 1
  END IF

LOOP UNTIL Count = WordNum
Slen = INSTR(Start, Source$, " ") - Start
END SUB

SUB FindCombKey (WordList$(), NumWordX, NumKeyWordFoundX, SentNum, NumKey, CombList() AS WordCode) STATIC
IF SentNum = 1 THEN
        KeyWordFoundNdx(SentNum).First = 1
ELSE
        ii = 1
       DO WHILE KeyWordFoundNdx(SentNum - ii).Last = 0 AND ii < SentNum - 1
        ii = ii + 1
        LOOP
        KeyWordFoundNdx(SentNum).First = KeyWordFoundNdx(SentNum - ii).Last + 1
END IF
NumKeyWordFound = 0
IF NumComb = 0 THEN EXIT SUB
FOR i = 1 TO NumWord' number of words to process

        '—— make lower case since combined keywords ignore case
        Word$ = LCASE$(WordList$(i))

        IF Word$ = "" GOTO SkipCombKey
            IF (ASC(Word$) >= 48 AND ASC(Word$) <= 57) GOTO SkipCombKey

'*********************************************************
'== Binary searches thru an CombList for a range for first letter

Last = 0
w$ = LCASE$(LEFT$(Word$, 1))
L = 1: R = NumCombX ' total number of code entries

DO

        x = (CLNG(L) + R) \ 2
IF w$ < LCASE$(LEFT$(CombList(x).Str, 1)) THEN
            R = x - 1
        ELSE
            IF w$ > LCASE$(LEFT$(CombList(x).Str, 1)) THEN
                L = x + 1
            ELSE   'if equal, check every other word for first and last
              M = 2
              IF x > 2 THEN
                  DO
                    IF LCASE$(LEFT$(CombList(x - M).Str, 1)) < w$ THEN
                          IF LCASE$(LEFT$(CombList(x - M + 1).Str, 1)) < w$ THEN
                                First = x - M + 2
                          ELSE
                                First = x - M + 1
                          END IF
                          EXIT DO
                    ELSE
                          M = M + 2
                          IF x - M <= 0 THEN First = 1: EXIT DO
                    END IF
                  LOOP
              ELSE
                    First = 1
              END IF
              IF x <= NumComb - 2 THEN
                    M = 2
                    DO
                        IF LCASE$(LEFT$(CombList(x + M).Str, 1)) > w$ THEN
                              IF LCASE$(LEFT$(CombList(x + M - 1).Str, 1)) > w$ THEN
                                    Last = x + M - 2
                              ELSE
                                    Last = x + M - 1
                              END IF
                              EXIT DO
                        ELSE
                              M = M + 2
                        IF x + M > NumComb THEN Last = NumComb: EXIT DO
                        END IF
                    LOOP
              ELSE
                  Last = NumComb
              END IF
            END IF
        END IF
LOOP UNTIL w$ = LCASE$(LEFT$(CombList(x).Str, 1)) OR L > R

  IF Last THEN

'*********************************************************

        '—— if it's a valid range, then do comparisons for words in the range
```

```
FOR j = Last TO First STEP -1
        Words = InCount(QPRTrim$(CombList(j).Str), " ") + 1'count number of words

        '---- if the keyword has more words than are left in the word list
        '     skip it, because there's no possibility of a match.
        IF Words > NumWord - i + 1 GOTO SkipCombKey

        CurrKey$ = ExtractWord$(CombList(j).Str, " ", 1)'extract first word
        Slen = LEN(CurrKey$)                            'of combined keyword
        Start = Slen + 1
        IF RIGHT$(CurrKey$, 1) = "/" THEN
                Exact = TRUE
                CurrKey$ = LEFT$(CurrKey$, Slen - 1)
                Slen = Slen - 1
        ELSE
                Exact = FALSE
        END IF

        '---- compare first word of combined key [CurrKey$]
        '     against the current document word [Word$]

        IF RIGHT$(Word$, 1) = "/" THEN Word$ = LEFT$(Word$, LEN(Word$) - 1)

        IF Exact THEN ' check for "exact" match
                Match = (LCASE$(CurrKey$) = LCASE$(Word$))
        ELSE
                Match = (LCASE$(CurrKey$) = LCASE$(LEFT$(Word$, Slen)))
        END IF

        ' no match, skip to next combined key in the First-Last range
        IF NOT Match GOTO SkipCombKey

        ' continue matching the rest of the words in the combined key
        ' exiting out as soon as there's a non-match
        AtFlag = FALSE
        NotFlag = FALSE

        FOR k = 1 TO Words - 1' number of words left in combined key

                ' extract the next word from the current combined keyword (j)
                CurrKey$ = ExtractWord$(CombList(j).Str, " ", Start)
                Slen = LEN(CurrKey$)
                Start = Start + Slen + 1

                IF RIGHT$(CurrKey$, 1) = "/" THEN
                        Exact = TRUE
                        CurrKey$ = LEFT$(CurrKey$, Slen - 1)
                        Slen = Slen - 1
                ELSE
                        Exact = FALSE
                END IF
                IF AtFlag = FALSE AND NotFlag = FALSE THEN
                        DocWord$ = WordList$(i + k)' Document word to compare
                ELSE
                        IF AtFlag = FALSE AND NotFlag = TRUE THEN
                                IF k < Words - 1 THEN DocWord$ = WordList$(i + k + 1)  'next word
                        ELSE
                                IF AtFlag = TRUE AND NotFlag = FALSE THEN
                                        DocWord$ = WordList$(i + k - 1) 'previous word
                                 ELSE
                                        DocWord$ = WordList$(i + k)
                                 END IF
                        END IF
                END IF
                IF RIGHT$(DocWord$, 1) = "/" THEN DocWord$ = LEFT$(DocWord$, LEN(DocWord$) - 1)

                IF CurrKey$ = "@" THEN ' special processing for @ wildcard
                        IF INSTR(AtList$, "/" + DocWord$ + "/") THEN
                                Match = TRUE' the word was in the @ list, so continue
                        ELSE
                                IF Lang$ = "GERMAN" THEN
                                        Match = TRUE
                                        AtFlag = TRUE
                                ELSE
                                        Match = FALSE
                                END IF
                        END IF
                        IF Match THEN

                                IF k < Words - 1 THEN
                                        DocWord$ = WordList$(i + k + 1)
                                        IF DocWord$ = "not" OR DocWord$ = "be" OR DocWord$ = "nicht" THEN NotFlag = TRUE
                                        END IF
                                END IF
                        ELSE
                                IF Exact THEN ' check for "exact" match
                                        Match = (LCASE$(CurrKey$) = LCASE$(DocWord$))
                                ELSE ' wildcard match, only compare # of chars in CurrKey$
                                        Match = (LCASE$(CurrKey$) = LCASE$(LEFT$(DocWord$, Slen)))
                                END IF
                        END IF

                        IF NOT Match THEN EXIT FOR

                NEXT' word in current combined keyword

                IF Match THEN ' this is a combined keyword, so add it to the list
                        NumKeyWordFound = NumKeyWordFound + 1
                        NumKey = NumKey + 1
                        REDIM PRESERVE KeyWordFound(1 TO NumKey) AS ListType
                        KeyWordFound(NumKey).Code = CombList(j).Code
```

```
                DO
                        SlashLoc = INSTR(CombStr$, "/")
                        IF SlashLoc THEN MID$(CombStr$, SlashLoc, 1) = " "
                LOOP UNTIL SlashLoc = 0
                KeyWordFound(NumKey).Num = i
                '-- blank out combined word from list so that single keys
                '   are not generated from parts of combined keys found
                TempStr$ = ""
                FOR k = i TO i + Words - 1
                        DO
                                IF RIGHT$(WordList$(k), 1) = "/" THEN
                                        WordList$(k) = LEFT$(WordList$(k), LEN(WordList$(k)) - 1)
                                ELSE
                                        EXIT DO
                                END IF
                        LOOP
                        TempStr$ = TempStr$ + WordList$(k) + " "
                        WordList$(k) = ""
                NEXT
                KeyWordFound(NumKey).Str = RTRIM$(TempStr$)
                EXIT FOR

        END IF


SkipCombKey:
        NEXT
END IF

NEXT' key in list

END SUB

SUB FindSingKey (WordList$(), NumWordX, NumKeyWordFoundX, SentNum, NumKey, SingList() AS WordCode) STATIC

'ARRAY NAME      LEN DESCRIPTION              DIRECTION  MODIFIED?
'----------      --- -----------------------  ---------  ---------
'SingList()      VAR  Single Keyword List     (Shared)   (Unchanged)
'KeyWordFound()  VAR  Single Keywords Found   (Returned) (Changed)
'WordList$()     VAR  Document Words          (Passed)   (Unchanged)

IF NumSing = 0 THEN EXIT SUB
FOR i = 1 TO NumWord' number of words in document

        Word$ = WordList$(i)

        IF Word$ = "" GOTO SkipSingKey
        IF (ASC(Word$) >= 48 AND ASC(Word$) <= 57) GOTO SkipSingKey
        PrefixFlag = FALSE: MeanPrefixFlag = FALSE
'*************************************************************
'== Binary searches thru an SingList for a range for first letter

TryAgain:
Last = 0
w$ = LCASE$(LEFT$(Word$, 1))
l = 1: R = NumSingX ' total number of code entries

DO

        x = (CLNG(l) + R) \ 2
IF w$ < LCASE$(LEFT$(SingList(x).Str, 1)) THEN
                R = x - 1
        ELSE
                IF w$ > LCASE$(LEFT$(SingList(x).Str, 1)) THEN
                        l = x + 1
                ELSE   'if equal, check every other word for first and last
                  M = 2
                  IF x > 2 THEN
                        DO
                          IF LCASE$(LEFT$(SingList(x - M).Str, 1)) < w$ THEN
                                  IF LCASE$(LEFT$(SingList(x - M + 1).Str, 1)) < w$ THEN
                                          First = x - M + 2
                                  ELSE
                                          First = x - M + 1
                                  END IF
                                  EXIT DO
                          ELSE
                                  M = M + 2
                                  IF x - M <= 0 THEN First = 1: EXIT DO
                          END IF
                        LOOP
                  ELSE
                          First = 1
                  END IF
                  IF x <= NumSing - 2 THEN
                          M = 2
                          DO
                                IF LCASE$(LEFT$(SingList(x + M).Str, 1)) > w$ THEN
                                        IF LCASE$(LEFT$(SingList(x + M - 1).Str, 1)) > w$ THEN
                                                Last = x + M - 2
                                        ELSE
                                                Last = x + M - 1
                                        END IF
                                        EXIT DO
                                ELSE
                                        M = M + 2
                                IF x + M > NumSing THEN Last = NumSing: EXIT DO
                                END IF
                          LOOP
                  ELSE
                          Last = NumSing
```

```
                END IF
              END IF
        END IF
LOOP UNTIL w$ = LCASE$(LEFT$(SingList(x).Str, 1)) OR L > R
IF Last THEN
        FOR j = Last TO First STEP -1
              CurrKey$ = QPRTrim$(SingList(j).Str)
              Slen = LEN(CurrKey$)
              IF RIGHT$(CurrKey$, 1) = "/" THEN
                    Exact = TRUE
                    CurrKey$ = LEFT$(CurrKey$, Slen - 1)
                    Slen = Slen - 1
              ELSE
                    Exact = FALSE
              END IF
              'compare the single keyword [CurrKey$/SingList(j).Str]
              'against the document word [Word$]
              IF Exact THEN ' check for "exact" match
                    Match = (CurrKey$ = Word$)
              ELSE ' check for wildcard match
                    Match = (CurrKey$ = LEFT$(Word$, Slen))
              END IF
          IF Match THEN ' add the single keyword to the list
              NumKeyWordFound = NumKeyWordFound + 1
              NumKey = NumKey + 1
              REDIM PRESERVE KeyWordFound(1 TO NumKey) AS ListType
              KeyWordFound(NumKey).Code = SingList(j).Code
              KeyWordFound(NumKey).Str = Word$
              KeyWordFound(NumKey).Num = i
              EXIT FOR
          END IF

NEXT' key in range
ELSE
        Letter$ = LEFT$(Word$, 2)
        IF Letter$ = "zs" OR Letter$ = "za" THEN
              SecNum = QPVALL$(RIGHT$(Word$, LEN(Word$) - 2))
              IF SecNum > 0 AND SecNum <= 3000 THEN
                    IF Letter$ = "zs" THEN
                          NumKeyWordFound = NumKeyWordFound + 1
                          NumKey = NumKey + 1
                          REDIM PRESERVE KeyWordFound(1 TO NumKey) AS ListType
                          KeyWordFound(NumKey).Code = SecNum + SecCode    '10563
                          KeyWordFound(NumKey).Str = "Sec" + STR$(SecNum)
                          KeyWordFound(NumKey).Num = i
                    ELSE
                          NumKeyWordFound = NumKeyWordFound + 1
                          NumKey = NumKey + 1
                          REDIM PRESERVE KeyWordFound(1 TO NumKey) AS ListType
                          IF SecNum <= 30 THEN
                                KeyWordFound(NumKey).Code = SecNum + ArtCode '13563
                                KeyWordFound(NumKey).Str = "Art" + STR$(SecNum)
                                KeyWordFound(NumKey).Num = i
                          END IF
                    END IF
              END IF
        END IF
END IF
IF NOT MeanPrefixFlag THEN
'check for meaningful prefixes. If found, divide word in two parts
Word$ = LCASE$(Word$)
LenW = LEN(Word$)
FOR NumLet = 14 TO 3 STEP -1
        IF LenW > NumLet + 3 THEN  'should leave at least 3 letters
              IF INSTR(MeanPrefixes$(NumLet), "\" + LEFT$(Word$, NumLet) + "\") THEN
                    WordTemp1$ = MID$(Word$, NumLet + 1)
                    Word$ = LEFT$(Word$, NumLet)
                    MeanPrefixFlag = TRUE
                    PrevMatch = Match 'save, because Match will change for the pref.
                    EXIT FOR
              END IF
        END IF
NEXT
IF MeanPrefixFlag THEN GOTO TryAgain 'check again
ELSE
        IF WordTemp1$ <> "" THEN
              IF PrevMatch THEN
                    Limit = 9
              ELSE
                    Limit = 6
              END IF
                    Word$ = WordTemp1$
                    WordTemp1$ = ""
                    IF LEN(Word$) >= Limit THEN GOTO TryAgain
        END IF
END IF
'check for meaningless prefixes and delete it
IF NOT PrefixFlag AND NOT Match THEN  'only one time
        Word$ = LCASE$(Word$)
        LenW = LEN(Word$)
        FOR NumLet = 8 TO 2 STEP -1
         IF LenW > NumLet + 3 THEN  'should leave at least 3 letters
              IF INSTR(Prefixes$(NumLet), "\" + LEFT$(Word$, NumLet) + "\") THEN
                    Word$ = MID$(Word$, NumLet + 1)
                    PrefixFlag = TRUE
                    EXIT FOR
              END IF
         END IF
        NEXT
```

```
        IF PrefixFlag <> EN
                Limit = 6
                IF LEN(Word$) >= Limit THEN GOTO TryAgain
        END IF
END IF

SkipSingKey:

NEXT' word in document
KeyWordFoundNdx(SentNum).Last = NumKey
END SUB

SUB GetKVList (DocNum&, SingList() AS WordCode, CombList() AS WordCode, KeyTemp AS KeyNdx127) STATIC

        KeyLen = LEN(KeyTemp)
'EMSGet KeyTemp, KeyLen, DocNum&, KeyEmsX
        FGetRT KeyEMS, KeyTemp, DocNum&, KeyLen
        '-- get only first 27% keywords and store them with synonims in RAM
        IF KeyTemp.Num > 10 THEN
                KeepNum = KeyTemp.Num * .27
                IF KeepNum < 7 THEN KeepNum = 7
        ELSE
                KeepNum = KeyTemp.Num
        END IF

        NumComb = 0: NumSing = 0 'both are common shared
        CALL DictSortSearch(KeyTemp, KeepNum, SingList(), CombList())
END SUB

FUNCTION GetSentValue! (SentNum, NumWord, Value AS WeightAvgNdx127, KeyTemp AS KeyNdx127) STATIC
'*** Value =SUMofUniqueKW(SV*PV^1/8) /SQR(TotalNumberOfWords)
        SentVal! = 0
        OldStr$ = ""
        FOR i = KeyWordFoundNdx(SentNum).First TO KeyWordFoundNdx(SentNum).Last
                'don't take word which is chopped already
                IF KeyWordFound(i).Num <> 0 THEN
                        'don't take the same word twice
                        IF INSTR(OldStr$, STR$(KeyWordFound(i).Code)) = 0 THEN
                                'find the word
                                FOR j = 1 TO KeyTemp.Num
                                        IF KeyTemp.Code(j) = KeyWordFound(i).Code THEN
                                                SentVal = SentVal + Value.Weight(j)
                                                OldStr$ = OldStr$ + STR$(KeyTemp.Code(j))
                                                EXIT FOR
                                        END IF
                                NEXT
                        END IF
                END IF
        NEXT
        GetSentValue! = SentVal! / SQR(NumWord)

END FUNCTION

FUNCTION InstrTblX (Start%, Source$, Chars$) STATIC

  'returns position first met char from chars$ in Source$
  VX = 0
  FOR i = 1 TO LEN(Chars$) STEP 2
        VX = INSTR(Start%, Source$, MID$(Chars$, i, 2))
        IF VX > 0 THEN EXIT FOR
  NEXT
  InstrTblX = VX
END FUNCTION

SUB LoadData STATIC
' Load in Word Sorted Dictionary directly into EMS (translate Code# to Word$)

EmsPg = EmsGetPFSegX

DIM DictWrdTemp AS DictType
File$ = LstDir$ + "DICT.Wrd" 'to get SecCode only
IF NOT ExistX(File$) THEN CLS : PRINT File$; " not found.": END
SizeofFile& = FileSize&(File$)
DictWrdNum = SizeofFile& \ LEN(DictWrdTemp)
SecCode = DictWrdNum - 3030 + 1
ArtCode = DictWrdNum - 30 + 1

File$ = LstDir$ + "DICTSORT.COD"
DIM DictCodeTemp AS DictType
IF NOT ExistX(File$) THEN CLS : PRINT File$; " not found.": END
SizeofFile& = FileSize&(File$)
DictCodeNum = SizeofFile& \ LEN(DictCodeTemp)
NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, DictCodeH%, File$
'IF EmsErrorX GOTO EMSErrHandler

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)

FOpen$ File$, DictCodeFILE
FOR i = 1 TO Num32kBlocks + 1

        BoxO 11, 5, 15, 70, 2, RevAttr
        PaintBoxO 11, 5, 15, 70, RevAttr
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1), 13, 12, RevAttr

        '-- map pages of the DictCodeH% memory to the EMS upper mem page frame

        FOR j = 1 TO 2
                EmsMapMem DictCodeH%, j, (i - 1) * 2 + j
                IF EmsErrorX GOTO EMSErrHandler
```

```
        NEXT

        '-- seek to beginning of current block
        FSeek DictCodeFILE, (i - 1) * ThirtyTwoK&

        IF i < Num32kBlocks + 1 THEN
                '-- get the 32k block and put it directly into the EMS page frame
                FGetA DictCodeFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&

        ELSE
                '-- load the left over (<32k) bytes
                FGetA DictCodeFILE, BYVAL EmsPg, BYVAL 0, LeftOver&

        END IF

NEXT

FClose DictCodeFILE
CLS
'---- Load Weight.Ndx into EMS
File$ = NdxDir$ + "NEWVAL.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
FOpenAll File$, 0, 4, ValueEMS
'ValueEMS = LoadIntoEMS(File$)
CLS
'---- Load Key.Ndx into EMS
File$ = NdxDir$ + "NEWKEY.NDX"
IF NOT Exist%(File$) THEN
        CLS
        PRINT File$; " not found."
        CALL ReleaseEMS
        STOP
END IF
FOpenAll File$, 0, 4, KeyEMS
'KeyEMS = LoadIntoEMS(File$)
END SUB

FUNCTION LoadIntoEMS (File$) STATIC

'-------- Returns the handle where the file was loaded into --------

EmsPg = EmsGetPFSeg%
SizeofFile& = FileSize&(File$)

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, FileEMS, File$

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
FOpenAll File$, 0, 4, LoadFILE
FOR i = 1 TO Num32kBlocks + 1
        Box0 11, 5, 15, 70, 2, RevAttr
        PaintBox0 11, 5, 15, 70, RevAttr
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks + 1), 13, 12, RevAttr

        '-- map pages of the EMS memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem FileEMS, j, (i - 1) * 2 + j
                IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP

        NEXT
        '-- seek to beginning of current block
        FSeek LoadFILE, (i - 1) * ThirtyTwoK&
        IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP
        IF i < Num32kBlocks + 1 THEN
                '-- get the 32k block and put it directly into the EMS page frame
                FGetA LoadFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&

                IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        ELSE
                '-- load the left over (<32k) bytes
                FGetA LoadFILE, BYVAL EmsPg, BYVAL 0, LeftOver&
                IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

        END IF
NEXT

FClose LoadFILE

CLS

LoadIntoEMS = FileEMS

END FUNCTION

SUB Rank (BestSent, NoMoreFlag, NumKeyWords(), NumWord(), Value AS WeightAvgNdx127, KeyTemp AS KeyNdx127) STATIC

'this procedure calculates value for each sentence and finds the best one.

REDIM SentValue(1 TO TotSentNum) AS SentValueType
FOR SentNum = 1 TO TotSentNum
        IF NumKeyWords(SentNum) > 0 THEN
                SentValue(SentNum).Value = GetSentValue!(SentNum, NumWord(SentNum), Value, KeyTemp)
```

```
        ELSE
                SentValue(SentNum).Value = 0
        END IF
                SentValue(SentNum).Num = SentNum
NEXT

'find the sentence with biggest value
k! = 0: Best = 0
FOR i = 1 TO TotSentNum
        IF SentValue(i).Value > k! THEN
        Best = i
        k! = SentValue(i).Value
        END IF
NEXT
IF Best = 0 THEN NoMoreFlag = TRUE: EXIT SUB
BestSent = SentValue(Best).Num
ERASE SentValue
END SUB

SUB ReadEnglishText (FirstLine&, LastLine&, HandleX, TotSentNumX) STATIC

        DIM Temp1  AS STR50
        REDIM SentNdx(1 TO 1) AS SentNdxType
        LenTemp = LEN(Temp1) ' = 80
        NumLines = 0   ' total number of lines input from file
        ActualLin = 0
        EndOfSentence$ = ". ! ? ; "
        LineNum = 0: SentNum = 1: SentNdx(1).BL = 1: SentNdx(1).BC = 1: s = 1
        EL = 0: EC = 0: BL = 0: BC = 0
        IF TextNum& > MaxText THEN
                FOR i& = FirstLine& TO LastLine&
                        FGetRT DocFile, Temp1, i&, LenTemp
                        CALL ReadSection(Temp1.Str, Section$(), Article$())
                        NumLines = NumLines + 1
                        EssSet1El Temp1, LenTemp, NumLines, Handle
                NEXT
        ELSE
                FOR i& = FirstLine& TO LastLine&
                        FGetRT DocFile, Temp1, i&, LenTemp
                        NumLines = NumLines + 1
                        TextArray(NumLines).Str = Temp1.Str
                        NEXT
                        Array2Ems TextArray(1), 128, NumLines, Handle
                FOR i = 1 TO NumLines
                        CALL ReadSection(TextArray(i).Str, Section$(), Article$())
                NEXT
        END IF
        '---- Process text
CurrLine = 0
GOSUB GetNextLine
DO
        DO
                '-- skip over blank lines, or if we've gone too far
                DO WHILE Start > LENTxt
                        IF CurrLine = NumLines GOTO EndOfFile
                        GOSUB GetNextLine
        ------------------
        LOOP

        p = InstrTblX(Start, Txt$, EndOfSentence$)
        IF p = 0 THEN
                IF CurrLine = NumLines GOTO EndOfFile
                GOSUB GetNextLine
                p = 0
        ELSE
                Start = p + 2
                EndSign$ = MID$(Txt$, p, 1)
        END IF
LOOP UNTIL p  '-- loop until we've found the end of sentence

IF p > 2 THEN '-- it's an end of sentence
-------------------------------' (i.e., can't have x. as the end of a sentence)
        IF MID$(Txt$, p - 2, 1) <> " " THEN '-- 2nd char before end of sentence location is not a space
                IF p + 2 < LENTxt THEN  '-- we're not at the end of the line, so check the next word/chars
                        IF MID$(Txt$, p + 2, 1) = " " THEN '-- we have two spaces after the end of sentence
                                '-- save potential end of sentence (line & column)
                                EL = LineNum: EC = p
                                '-- look for beginning of sentence and
                                '   the first alphanumeric character in that sentence
                                ch = MidChar(Txt$, p)
                                FirstFlag = TRUE
                                DO UNTIL (ch >= 97 AND ch <= 122) OR (ch >= 48 AND ch <= 57) OR (ch >= 65 AND ch <= 90
                                        p = p + 1
                                        IF ch <> 32 AND ch <> 0 AND FirstFlag THEN BC = p: BL = LineNum: FirstFlag =
                                                IF p > LENTxt THEN
                                                        IF CurrLine = NumLines GOTO EndOfFile
                                                        GOSUB GetNextLine
                                                END IF
                                        ch = MidChar(Txt$, p)
                                LOOP
                                IF (ch >= 65 AND ch <= 90) OR MidChar(Txt$, p + 1) = 46 OR MidChar(Txt$, p + 1) = 41 O
                                        '-- this is definitely a new sentence or sequential number
                                        ' so save location info
                                        SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = EC
                                        s = SentNum
                                        SentNum = SentNum + 1
                                        REDIM PRESERVE SentNdx(1 TO SentNum) AS SentNdxType
                                        IF TextNum& > MaxText THEN
                                                MID$(Txt$, p, 1) = LCASE$(MID$(Txt$, p, 1))
                                                Temp1.Str = Txt$
                                                EssSet1El Temp1, LenTemp, CurrLine, Handle
```

```
                                        ELSE
                                                MID$(TextArray(CurrLine).Str, p, 1) = LCASE$(MID$(TextArray(CurrLine).
                                        END IF
                                        SentNdx(SentNum).BL = BL: SentNdx(SentNum).BC = BC
                                        Start = p
                                END IF '-- end check for uppercase letter

                                '-- search for the next [potential] EOS no matter
                                '   if we found a sentence (above) or not
                                GOTO NextEnglSearch
                        END IF   '-- two spaces after EOS
                END IF   '-- not end of line

                '-- at this point we have: xx._x
                EL = LineNum: EC = p '-- save potential EOS info
                '-- this loop extracts the word to the left of the EOS
                n = QInstrB(p, Txt$, " ")
                '-- extract the previous word
                PrevWord$ = LCASE$(MID$(Txt$, n + 1, p - n - 1))
                '-- check to see if the previous is among the 8 abbreviations
                AbbrevEnglNum = 8
                CALL FindExact(VARPTR(AbbrevEngl$(1)), AbbrevEnglNum, PrevWord$)
                '-- if we found the abbreviation then this wasn't an EOS
                IF AbbrevEnglNum <> -1 THEN GOTO NextEnglSearch
                '-- wasn't found, move p to point to the space (after the EOS char)
                p = p + 1
                IF p > LENTxt THEN   '-- we were already at the end of the line
                                                        '   so get the next line of text
                        IF CurrLine = NumLines THEN
                                GOTO EndOffFile
                        ELSE
                                GOSUB GetNextLine
                        END IF
                END IF
                '[same loop as above]
                FirstFlag = TRUE
        DO
                ch = MidChar(Txt$, p)
                IF ch <> 32 AND ch <> 0 AND FirstFlag THEN BC = p: BL = LineNum: FirstFlag = FALSE
                p = p + 1
                IF p > LENTxt THEN
                        IF CurrLine = NumLines GOTO EndOffFile
                        GOSUB GetNextLine
                END IF
        LOOP UNTIL (ch >= 97 AND ch <= 122) OR (ch >= 48 AND ch <= 57) OR (ch >= 65 AND ch <= 90)
        '-- if it's not an uppercase letter, continue searching (this wasn't an EOS)
IF (ch >= 65 AND ch <= 90) OR MidChar(Txt$, p) = 46 OR MidChar(Txt$, p) = 41 OR EndSign$ = ";" THEN    '-- U
                '-- now we assume that it's an EOS, save info
                SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = EC
                s = SentNum
                SentNum = SentNum + 1
                REDIM PRESERVE SentNdx(1 TO SentNum) AS SentNdxType
                IF TextNum& > MaxText THEN
                        IF p > 1 THEN
                                MID$(Txt$, p - 1, 1) = LCASE$(MID$(Txt$, p - 1, 1))
                        ELSE
                                MID$(Txt$, p, 1) = LCASE$(MID$(Txt$, p, 1))
                        END IF
                        Temp1.Str = Txt$
                        EmsSet1El Temp1, LenTemp, CurrLine, Handle
                ELSE
                        IF p > 1 THEN
                                MID$(TextArray(CurrLine).Str, p - 1, 1) = LCASE$(MID$(TextArray(CurrLine).Str,
                        ELSE
                                MID$(TextArray(CurrLine).Str, p, 1) = LCASE$(MID$(TextArray(CurrLine).Str, p,
                        END IF
                END IF
                SentNdx(SentNum).BL = BL: SentNdx(SentNum).BC = BC
                Start = p
        END IF
    END IF
END IF
NextEnglSearch:
    LOOP UNTIL CurrLine >= NumLines

EndOffFile:
    SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = 80
    TotSentNum = SentNum
    IF TextNum& > MaxText THEN
        FOR i = 1 TO ActualLin
                EmsGet1El Temp1, LenTemp, i, Handle
                Txt$ = QPRTrim$(Temp1.Str)
                CALL ChangeChar(Txt$, Keep$)
                Temp1.Str = Txt$
                EmsSet1El Temp1, LenTemp, 1, Handle
        NEXT
    ELSE
        FOR i = 1 TO ActualLin
                CALL ChangeChar(TextArray(i).Str, Keep$)
        NEXT
    END IF
    EXIT SUB

'------------

GetNextLine:
    CurrLine = CurrLine + 1
    Start = 1   '-- start scanning at first position
    p = 1
```

```
        LineNum = LineNum + 1
        ActualLin = ActualLin + 1
        IF TextNum& > MaxText THEN
                EmsGet1El Temp1, LenTemp, CurrLine, Handle
                Txt$ = QPRTrim$(Temp1.Str) + " "
        ELSE
                Txt$ = QPRTrim$(TextArray(CurrLine).Str) + " "
        END IF
        '-- trim down end of line, but make sure there's one space at the end
        '   so that we can find end of sentences (looking for a DOT & SPACE)
        '   even if they're at the end of the line.
        LENTxt = LEN(Txt$)
        RETURN


END SUB
SUB ReadGermanText (FirstLine&, LastLine&, Handle%, Lin%) STATIC

        DIM Temp1  AS STR80
        REDIM SentNdx(1 TO 1) AS SentNdxType
        LenTemp = LEN(Temp1) ' = 80
        NumLines = 0    ' total number of lines input from file
        ActualLin = 0
        EndOfSentence$ = ". ! ? "
        LineNum = 0: SentNum = 1: SentNdx(1).BL = 1: SentNdx(1).BC = 1: s = 1
        IF TextNum& > MaxText THEN
                FOR i& = FirstLine& TO LastLine&
                        FGetRT DocFile, Temp1, i&, 80
                        CALL ReadSection(Temp1.Str, Paragraph$(), Artikel$())
                        NumLines = NumLines + 1
                        EmsSet1El Temp1, LenTemp, NumLines, Handle
                NEXT
        ELSE
                FOR i& = FirstLine& TO LastLine&
                        FGetRT DocFile, Temp1, i&, LenTemp
                        NumLines = NumLines + 1
                        TextArray(NumLines).Str = Temp1.Str
                NEXT
                        Array2Ems TextArray(1), 128, NumLines, Handle
                FOR i = 1 TO NumLines
                        CALL ReadSection(TextArray(i).Str, Paragraph$(), Artikel$())
                NEXT
        END IF

          '---- Process text
 CurrLine = 0
 GOSUB NextLine
 DO
                DO
                        '-- skip over blank lines, or if we've gone too far
                        DO WHILE Start > LENTxt
                                IF CurrLine = NumLines GOTO EndFile
                                GOSUB NextLine
                        LOOP

                        p = InstrTblX(Start, Txt$, EndOfSentence$)
                        IF p = 0 THEN
                                IF CurrLine = NumLines GOTO EndFile
                                GOSUB NextLine
                                p = 0
                        ELSE
                                Start = p + 2
                        END IF
                LOOP UNTIL p  '-- loop until we've found the end of sentence

                IF p > 2 THEN '-- it's an end of sentence
                                        ' (i.e., can't have x. as the end of a sentence)
                        '-- extract the previous word
                        n = QInstrB(p, Txt$, " ")
                        PrevWord$ = LCASE$(MID$(Txt$, n + 1, p - n - 1))

                        NOSFlag = FALSE
                        IF MID$(Txt$, p - 2, 1) = " " THEN NOSFlag = TRUE'-- 2nd char before end of sentence location is not a
                        IF p + 2 < LENTxt THEN  '-- we're not at the end of the line, so check the next word/chars
                                IF MID$(Txt$, p + 2, 1) = " " THEN '-- we have two spaces after the end of sentence
                                        '-- save potential end of sentence (line & column)
                                        EL = LineNum: EC = p
                                        '-- look for beginning of sentence and
                                        '   the first alphanumeric character in that sentence
                                        ch = MidChar(Txt$, p)
                                        FirstFlag = TRUE
                                        DO UNTIL (ch >= 97 AND ch <= 122) OR (ch >= 48 AND ch <= 57) OR (ch >= 65 AND ch <= 90
                                                p = p + 1
                                                IF ch <> 32 AND ch <> 0 AND FirstFlag THEN BC = p: BL = LineNum: FirstFlag =
                                                        IF p > LENTxt THEN
                                                                IF CurrLine = NumLines GOTO EndFile
                                                                GOSUB NextLine
                                                        END IF
                                                ch = MidChar(Txt$, p)
                                        LOOP
                                        IF ch >= 65 AND ch <= 90 THEN '-- Uppercase letter
                                                '-- this is definitely a new sentence, so save location info
                                                SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = EC
                                                s = SentNum
                                                SentNum = SentNum + 1
                                                REDIM PRESERVE SentNdx(1 TO SentNum) AS SentNdxType
                                                SentNdx(SentNum).BL = BL: SentNdx(SentNum).BC = BC
                                                Start = p
                                        END IF '-- end check for uppercase letter
```

```
                '-- search for the next [potential] EOS no matter
                '   if we found a sentence (above) or not
                GOTO NextSearch
            END IF   '-- two spaces after EOS
        END IF   '-- not end of line

        '-- at this point we have: x._x
        EL = LineNum: EC = p '-- save potential EOS info
        ' look for  the first alphanumeric character
        FirstFlag = TRUE
        DO
                p = p + 1
                ch = MidChar(Txt$, p)
                IF ch <> 32 AND ch <> -1 AND ch <> 0 AND FirstFlag THEN BC = p: BL = LineNum: FirstFlag = FALS
                IF p > LENTxt THEN
                        IF CurrLine = NumLines GOTO EndFile
                        GOSUB NextLine
                END IF
        LOOP UNTIL (ch >= 97 AND ch <= 122) OR (ch >= 48 AND ch <= 57) OR (ch >= 65 AND ch <= 90)
        '-- if it's not an uppercase letter, continue searching (this wasn't an EOS)
        IF ch < 65 OR ch > 90 THEN GOTO NextSearch
          '-- extract the word following the EOS
          n = p
        DO
          ch$ = MID$(Txt$, n, 1)
          n = n + 1
        LOOP UNTIL (ch$ < "A" OR ch$ > "Z") AND (ch$ < "a" OR ch$ > "z") OR n > LENTxt
        '-- "following word"
        FollWord$ = MID$(Txt$, p, n - p - 1)
        '-- check to see if this is one of the 316 noise words
        NoiseNum = 316
        CALL FindExact(VARPTR(Noise$(1)), NoiseNum, FollWord$)
        IF NoiseNum <> -1 THEN  '-- it was a noise word, so we know it's an EOS
                SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = EC
                s = SentNum
                SentNum = SentNum + 1
                REDIM PRESERVE SentNdx(1 TO SentNum) AS SentNdxType
                SentNdx(SentNum).BL = BL: SentNdx(SentNum).BC = BC
                Start = p
                GOTO NextSearch
        END IF
                IF NOSFlag THEN GOTO NextSearch
                '-- check to see if the previous is among the 15 abbreviations
                Abbrev1Num = 15
                CALL FindExact(VARPTR(Abbrev1$(1)), Abbrev1Num, PrevWord$)
                '-- if we found the abbreviation then this wasn't an EOS
                IF Abbrev1Num <> -1 THEN GOTO NextSearch
                '-- wasn't found, move p to point to the space (after the EOS char)

        '-- check to see if Length < 6 (which would not be a new sentence)
        TrySent$ = ""
        IF SentNdx(SentNum).BL = LineNum THEN  '-- sentence starts on current line
                TrySent$ = MID$(Txt$, SentNdx(SentNum).BC, EC - SentNdx(SentNum).BC + 1)
        ELSE
                IF SentNdx(SentNum).BL = LineNum - 1 THEN
                  TrySent$ = RIGHT$(PrevLine$, LEN(PrevLine$) - SentNdx(SentNum).BC + 1) + LEFT$(Txt$, EC)
                END IF
        END IF
        TLen = LEN(TrySent$)
        IF TLen THEN
                CALL CRUNCH(TrySent$, " ", TLen)
                TrySent$ = LEFT$(TrySent$, TLen)
                Num = InCount(TrySent$, " ") + 1
                IF Num < 6 THEN GOTO NextSearch  '-- Sentence has less than 6 words, so it wasn't an EOS
        END IF
        '-- now we assume that it's an EOS, save info
        SentNdx(SentNum).EL = EL: SentNdx(SentNum).EC = EC
        s = SentNum
        SentNum = SentNum + 1
        REDIM PRESERVE SentNdx(1 TO SentNum) AS SentNdxType
        SentNdx(SentNum).BL = BL: SentNdx(SentNum).BC = BC
        Start = p
END IF

NextSearch:

        LOOP UNTIL CurrLine >= NumLines

EndFile:
         SentNdx(SentNum).EL = TextNum&: SentNdx(SentNum).EC = 80
         TotSentNum = SentNum
         IF TextNum& > MaxText THEN
                FOR i = 1 TO ActualLin
                        EmsGet1El Temp1, LenTemp, i, Handle
                                CALL Lower(Temp1.Str)
                                Txt$ = QPRTrim$(Temp1.Str)
                                CALL ChangeChar(Txt$, Keep$)
                                Temp1.Str = Txt$
                                EmsSet1El Temp1, LenTemp, i, Handle
                NEXT
         ELSE
                FOR i = 1 TO ActualLin
                        CALL Lower(TextArray(i).Str)
                        CALL ChangeChar(TextArray(i).Str, Keep$)
                NEXT
         END IF
         EXIT SUB

'-----------
NextLine:
        CurrLine = CurrLine + 1
```

```
        Start = 1   '— start scanning at first position
        p = 1
        LineNum = LineNum + 1
        ActualLin = ActualLin + 1
        '— save the previous line for checking length of sentence
        '   (in the cases where a sentence crosses the line boundary)
        PrevLine$ = Txt$
        IF TextNum& > MaxText THEN
                EmsGet1Si Temp1, LenTemp, CurrLine, Handle
                Txt$ = QPRTrim$(Temp1.Str) + " "
        ELSE
                Txt$ = QPRTrim$(TextArray(CurrLine).Str) + " "
        END IF
        '— trim down end of line, but make sure there's one space at the end
        '   so that we can find end of sentences (looking for a DOT & SPACE)
        '   even if they're at the end of the line.
        LENTxt = LEN(Txt$)
        RETURN

END SUB

SUB ReadSection (Txt$, SecArray$(), ArtArray$()) STATIC
'—Look for "sections" or "articles"

IF Lang$ = "GERMAN" THEN
        '— In GERMAN it's Par
        SearchStr$ = "Par"
ELSE
        '— In English it's Sec
        SearchStr$ = "Sec"
END IF

Letter$ = "zs"

FOR LookStep = 1 TO 2

        Start = 1

        DO
                M = INSTR(Start, Txt$, SearchStr$)  '— column of start of Sec. or Art.

                IF M THEN

                        j = INSTR(M, Txt$, " ") '— position of the end of the word

                        IF j THEN   '— if this is not a last word
                                Word$ = MID$(Txt$, M, j - M)  '— get the whole word
                        ELSE
                                EXIT DO  '— this was the last word, so exit
                        END IF

                        '— check if the word matches variations on Section or Article
                        IF Lang$ = "GERMAN" AND LookStep = 1 THEN

ELSE
        NumFound = 3   '— there are three variations that we check for
END IF

IF LookStep = 1 THEN
        CALL FindExact(VARPTR(SecArray$(1)), NumFound, Word$)
ELSE
        CALL FindExact(VARPTR(ArtArray$(1)), NumFound, Word$)
END IF

IF NumFound <> -1 THEN  '— it did match, so check the number
        k = j + 1   '— starting position of [potential] number

        DO '— skip over blank spaces
                ch = MidChar(Txt$, k)
                k = k + 1
        LOOP UNTIL ch <> 32 OR k > LEN(Txt$)

        a1 = 0

        DO '— collect the whole number
                ch = MidChar(Txt$, k + a1 - 1)
                a1 = a1 + 1
                IF k + a1 - 1 > LEN(RTRIM$(Txt$)) THEN EXIT DO
        LOOP UNTIL ch < 48 OR ch > 57

        IF a1 > 1 THEN 'there is a number
                Numb$ = MID$(Txt$, k - 1, a1 - 1)
                IF QPVall&(Numb$) <= 3000 AND QPVall&(Numb$) > 0 THEN

                        '— if we're looking for Article numbers, don't accept
                        '   article numbers over 30
                        IF LookStep = 2 AND QPVall&(Numb$) > 30 THEN GOTO NextStep
                        NewWord$ = Letter$ + Numb$
                        a1 = INSTR(k, Txt$, " ")
                        IF a1 = 0 THEN
                                a1 = LEN(Txt$)
                        ELSE
                                NewWord$ = NewWord$ + STRING$(a1 - M - LEN(NewWord$), 32)
                        END IF
                        Txt$ = LEFT$(Txt$, M - 1) + NewWord$ + MID$(Txt$, a1)
                END IF
        ELSE
                IF SearchStr$ = "Art" THEN

                        a1 = 0
```

```
                '— loop while it's a Roman numeral and we're not
                '   past the end of the string
                DO
                        ch$ = MID$(Txt$, k + m1 - 1, 1)
                        m1 = m1 + 1
                        IF k + m1 - 1 > LEN(RTRIM$(Txt$)) THEN EXIT DO
                LOOP WHILE INSTR("IVX", ch$)

                Numb$ = MID$(Txt$, k - 1, m1 - 1)

                '— translate the Roman numeral(s) to Arabic numerals
                NumFound = 30  '— there are 30 Roman numbers to check
                CALL FindExact(VARPTR(Numbers$(1)), NumFound, Numb$)

                IF NumFound <> -1 THEN

                        NewWord$ = "za" + LTRIM$(STR$(NumFound + 1))
                        m1 = INSTR(k, Txt$, " ")
                        IF m1 = 0 THEN
                                Txt$ = LEFT$(Txt$, M - 1) + NewWord$
                        ELSE
                                NewWord$ = NewWord$ + STRING$(m1 - M - LEN(NewWord$), 32)
                                Txt$ = LEFT$(Txt$, M - 1) + NewWord$ + RIGHT$(Txt$, LEN(Txt$) - m1 + 1
                        END IF

                END IF

        END IF   '— are we searching for an Article #?

    END IF   '— there's a number after the Section/Article

END IF   '— did we find a variation of Section or Article?
        END IF   '— INSTR(Text,"Sec.") was found

NextStep:

        Start = M + 1

    LOOP UNTIL M = 0

    SearchStr$ = "Art"
    Letter$ = "za"

NEXT

'— start looking at the beginning of the line
Start = 1

DO
    '— look for the section symbol
    M = INSTR(Start, Txt$, CHR$(21))

    '— if we found one, process it
    IF M THEN

        '— position right after the symbol
        k = M + 1
        m1 = 0

        '— loop until it's not a number (a space is ok, however)
        '   or we've reached the end of the string
        DO
            ch = MidChar(Txt$, k + m1)
            m1 = m1 + 1
            IF c + m1 - 1 > LEN(RTRIM$(Txt$)) THEN EXIT DO
        LOOP UNTIL (ch < ASC0 OR ch > ASC9) AND ch <> 32

        '— the number is the position from right after the symbol (k)
        '   to the non-number position found in the loop above (m1 - 1)
        Numb$ = QPTrim$(MID$(Txt$, k, m1 - 1))

        IF QPVall&(Numb$) <= 3000 AND QPVall&(Numb$) > 0 THEN

            NewWord$ = "zs" + Numb$
            m1 = INSTR(k + 1, Txt$, " ")

            IF m1 THEN
                Txt$ = LEFT$(Txt$, M - 1) + NewWord$ + MID$(Txt$, m1)
            ELSE
                Txt$ = LEFT$(Txt$, M - 1) + NewWord$
            END IF

        END IF

        '— start looking at the next position
        Start = M + 2

    END IF

'— loop until we don't find any more section symbols
LOOP UNTIL M = 0

END SUB

'-----------------------------------------------------------------
SUB ReleaseEMS STATIC

IF KeyEMS THEN EmsRelMem KeyEMS
IF ValueEMS THEN EmsRelMem ValueEMS
IF DictCodeH THEN EmsRelMem DictCodeHX
```

```
END SUB

SUB WordParse (Sent$, LenSen, SentNum, WordList$(), Words%) STATIC

'---------- Parse Sentence into Words
'--Words=number of words parsed

        Words = 0: Slen = 0: Start = 1
        DO UNTIL Start >= LenSen
              Slen = 0
              FOR i = Start TO LenSen
                      ch$ = MID$(Sent$, i, 1)
                      IF ch$ = " " THEN
                              EXIT FOR
                      ELSE
                              Slen = Slen + 1
                      END IF
              NEXT
              IF Slen > 0 THEN
              '------ extract word
                     w$ = MID$(Sent$, Start, Slen)
                     '------ fill out 1 and 2 char words with /'s
                     IF Slen < 3 THEN w$ = w$ + STRING$(3 - Slen, "/")
                     '------ allow only words that start with alphabetic chars "a"-"z" or 1-9
                     ASCw = ASC(w$)
                     IF (ASCw >= 97 AND ASCw <= 122) OR (ASCw >= 65 AND ASCw <= 90) THEN
                            Words = Words + 1
                            '------ the following doesn't apply to GERMAN
                            IF Lang$ = "ENGLISH" THEN
                                   IF RIGHT$(w$, 2) = "'s" THEN     '-- remove the 's
                                          w$ = LEFT$(w$, Slen - 2)
                                   ELSEIF RIGHT$(w$, 1) = "'" THEN  '-- and any final '
                                          w$ = LEFT$(w$, Slen - 1)
                                   END IF
                            END IF

                       '------ store the word
                       REDIM PRESERVE WordList$(1 TO Words)
                       WordList$(Words) = w$
                     END IF
        END IF
 Start = Start + Slen + 1
 LOOP 'next word in sentence

END SUB

SUB WriteSentence (Sent$, Best, AbstrPos&) STATIC

' this procedure wraps sentence, saves highligting information and write
' sentence to the disk. Marking EOS - add 100 to the position of the first
' highlighted word.

       Wid = 78: OldStr$ = ""
       'create array of words which should be hihglighted
        n = 0
        'move numbers into array
        NumCodes = KeyWordFoundNdx(Best).Last - KeyWordFoundNdx(Best).First + 1
        REDIM KeyWords$(1 TO NumCodes)
        FOR i = KeyWordFoundNdx(Best).First TO KeyWordFoundNdx(Best).Last
        ' to not highlight word, appeared in the previous sentence take out this
        IF KeyWordFound(i).Num <> 0 THEN                          ' comments.
                       IF INSTR(OldStr$, RTRIM$(KeyWordFound(i).Str) + " ") = 0 THEN
                              n = n + 1
                              KeyWords$(n) = RTRIM$(KeyWordFound(i).Str)
                              OldStr$ = OldStr$ + KeyWords$(n) + " "
          END IF
             END IF
        NEXT
        REDIM PRESERVE KeyWords$(1 TO n)

        'collecting highlighting information
        SentClean$ = Sent$
        CALL ChangeChar(SentClean$, .KeepNoNumbers$)
        IF Lang$ = "GERMAN" THEN CALL Lower(SentClean$)
        NumHigh = 0
        REDIM Highligt(1 TO NumCodes)
        REDIM LenHigh(1 TO NumCodes)
        FirstTime = TRUE
        FOR i = 1 TO UBOUND(KeyWords$)
                Start = 1
                DO
                       NumHigh = NumHigh + 1
                       IF Lang$ = "GERMAN" THEN
                              Highligt(NumHigh) = INSTR(Start, SentClean$, KeyWords$(i) + " ")
                       ELSE
                              IF FirstTime THEN
                                     Highligt(NumHigh) = INSTR(SentClean$, UCASE$(LEFT$(KeyWords$(i), 1)) + MID$(KeyWords$(i), 2)
                              END IF
                              Highligt(NumHigh) = INSTR(Start, SentClean$, KeyWords$(i) + " ")
                       END IF
             END IF
             LenHigh(NumHigh) = LEN(KeyWords$(i))
             'if still not found try add "'s"
             IF Highligt(NumHigh) = 0 THEN
                    SpLoc = INSTR(KeyWords$(i), " ")
                    IF SpLoc THEN
                           KW$ = LEFT$(KeyWords$(i), SpLoc - 1) + "'s" + MID$(KeyWords$(i), SpLoc)
                    ELSE
                           KW$ = KeyWords$(i) + "'s"
```

```
                END IF
                Highligt(NumHigh) = INSTR(Start, SentClean$, KW$)
                LenHigh(NumHigh) = LEN(KW$)
            END IF

            FirstTime = FALSE
            IF Highligt(NumHigh) <> 0 THEN
                MID$(SentClean$, Highligt(NumHigh), LenHigh(NumHigh)) = STRING$(LenHigh(NumHigh), 32)
            END IF
            Start = Highligt(NumHigh) + LenHigh(NumHigh)
        LOOP UNTIL Highligt(NumHigh) = 0 OR NumHigh = NumCodes OR Start >= LEN(Sent$)
        IF Highligt(NumHigh) = 0 THEN NumHigh = NumHigh - 1
        IF NumHigh = NumCodes THEN EXIT FOR
NEXT

'wrapping
DIM AbstrLine AS AbstrType
DO
  IF LEN(Sent$) > Wid THEN
      LastSpc = CInstrBX(Wid + 1, Sent$, " ")
      IF LastSpc <> 0 THEN
          Text$ = RTRIM$(LEFT$(Sent$, LastSpc))
          '-- remove portion of string that's been moved to the Text$
          Sent$ = MID$(Sent$, LastSpc + 1)
          j = 0
          'save highlighting information
          FOR i = 1 TO NumHigh
              IF Highligt(i) <> 0 THEN
                  IF Highligt(i) <= LastSpc THEN 'save it
                      j = j + 1
                      IF j > 5 THEN EXIT FOR
                      AbstrLine.Word(j) = CHR$(Highligt(i))
                      IF Highligt(i) + LenHigh(i) > LastSpc THEN
                          'probably it is comb. kw which was wrapped
                          AbstrLine.Lenth(j) = CHR$(LastSpc - Highligt(i))
                          LenHigh(i) = LenHigh(i) - (LastSpc - Highligt(i) + 1)
                          Highligt(i) = 1
                      ELSE
                          AbstrLine.Lenth(j) = CHR$(LenHigh(i))
                          Highligt(i) = 0
                      END IF
                  ELSE
                      Highligt(i) = Highligt(i) - LastSpc 'substract the lenth of line
                  END IF
              END IF
          NEXT
      ELSE
          Text$ = RTRIM$(LEFT$(Sent$, Wid))
          Sent$ = MID$(Sent$, Wid + 1)
      END IF
      FOR k = j + 1 TO 5
          AbstrLine.Word(k) = CHR$(0)
          AbstrLine.Lenth(k) = CHR$(0)
      NEXT

      IF LEN(Sent$) = 0 THEN
          'mark EOS
          AbstrLine.Word(1) = CHR$(ASC(AbstrLine.Word(1)) + 100)
      END IF
      AbstrLine.Str = Text$
      'PRINT #9, Text$
      FPutRT AbstrFile, AbstrLine, AbstrPos&, 88
      CALL DosErrHandler
      AbstrPos& = AbstrPos& + 1
  END IF
LOOP WHILE LEN(Sent$) > Wid
IF LEN(Sent$) <> 0 THEN
      Text$ = RTRIM$(Sent$)
      j = 0
          'save highlighting information
          FOR i = 1 TO NumHigh
              IF Highligt(i) <> 0 THEN
                  j = j + 1
                  IF j > 5 THEN EXIT FOR
                  AbstrLine.Word(j) = CHR$(Highligt(i))
                  AbstrLine.Lenth(j) = CHR$(LenHigh(i))
              END IF
          NEXT
          FOR k = j + 1 TO 5
              AbstrLine.Word(k) = CHR$(0)
              AbstrLine.Lenth(k) = CHR$(0)
          NEXT
          AbstrLine.Word(1) = CHR$(ASC(AbstrLine.Word(1)) + 100)
'PRINT #9, Text$
          AbstrLine.Str = Text$
          FPutRT AbstrFile, AbstrLine, AbstrPos&, 88
          CALL DosErrHandler
          AbstrPos& = AbstrPos& + 1
      END IF
END SUB

'this program sanitizes both english and german abstract

'If you are going to try to understand this program then
'you need at least one bottle of vodka.
'I feel sorry for you. Go ahead.

DEFINT A-Z
TYPE AbstrSanType
        Str AS STRING * 78
        Rest AS STRING * 6
END TYPE
```

```
CONST FALSE = 0, TRUE = NOT FALSE, Sing = 0, Comb = NOT Sing

'$INCLUDE: '\user\include\types.bi'
'$INCLUDE: '\user\include\declares.bi'

DECLARE SUB FindSingKey (Word$; KeyEMS%, NewSentFlag, SingFlag, Limit)
DECLARE SUB FindCombKey (WordPhrase$(), KeyEMS%, CombFlag)
DECLARE SUB EmsAlloc (NumPages%, Handle%, LoadFILE$)
DECLARE SUB InsertStr (BYVAL Address%, Inserted$, Size%)
DECLARE SUB Config ()
DECLARE SUB LoadData ()
DECLARE SUB Wrapping (Sent$, Text$(), NumLines%)

DECLARE FUNCTION LoadIntoEMS% (File$)
DECLARE FUNCTION FirstLast% (Word$, First%, Last%, KeyType%)
DECLARE FUNCTION NotInstr (Start%, Searched$, Table$)

COMMON SHARED Fg, Bg, Error, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
COMMON SHARED SixteenK, SixtyFour, ThirtyTwo, ThirtyTwoK$
COMMON SHARED Beg, Fin, Machine$, Noise$(), SingKeywordEMS%, CombKeyWordEMS%
COMMON SHARED SingTable%(), CombTable%(), XLateTable%(), NextStart, Articles$

x$ = ""

CALL Config
CALL LoadData

FileName$ = AbstrDir$ + ".NDX"
FOpenAll FileName$, 2, 4, AbstrNdxFile
FileName$ = AbstrDir$ + ".txt"
FOpenAll FileName$, 2, 4, AbstrFile
'File$ = AbstrDir$ + ".TX" + Machine$
'OPEN File$ FOR OUTPUT AS #1
DIM AbstrNdx AS ISAMtype
LenNdx = LEN(AbstrNdx)
DIM Txt AS AbstrSanType
LenTxt = LEN(Txt)
NoFirst$ = CHR$(0) + " '({[" + CHR$(34)
UpTable$ = "QWERTYUIOPLKJHGFDSAZXCVBNM"

REDIM GermanNotName$(3 TO 9)
FOR i = 3 TO 9
        READ GermanNotName$(i)
NEXT
REDIM EnglishNotName$(2 TO 6)
FOR i = 2 TO 6
        READ EnglishNotName$(i)
NEXT
GermanData:
DATA "\ion\ung\","\heit\nzip\satz\keit\","\ionen\nzips\ungen\viele\"
DATA "\kosten\schaft\heiten\keiten\manche\einige\vielen\saetze\satzes\","\pflicht\saetzen\manchen\einigen\mehrere\"
DATA "\schaften\mehreren\","\pflichten\"

EnglishData:
DATA "\al\","\ion\ity\als\","\ment\ions\ency\ancy\ness\","\ments\ional\ities\"
DATA "\mental\encies\ancies\"

REDIM GermanName$(1 TO 2)
FOR i = 1 TO 2
        READ GermanName$(i)
NEXT
DATA "\er\ow\","\mann\burg\berg\"

REDIM EnglishName$(1 TO 4)
FOR i = 1 TO 4
        READ EnglishName$(i)
NEXT
DATA "\o\i\","\er\rg\os\ey\tz\","\man\son\ong\haa\ton\scn\","\mann\tein\"

a$ = "\ein\eine\einem\einen\eines\diese\dieses\dieser\diesen\diesem\dieser"
b$ = "\kein\keine\keinem\keinen\keinem\keines\"
eng$ = "\a\an\any\this\such\no\several\many\"
IF Lang$ = "GERMAN" THEN
        Articles1$ = a$ + b$
        Articles2$ = "\der\die\das\den\dem\des\"
ELSE
        Articles1$ = eng$
        Articles2$ = "\the\"
END IF

IF Lang$ = "GERMAN" THEN
        SectionLine$ = "\Artikel\Art\Art.\Paragraph\Par\Par.\"
ELSE
        SectionLine$ = "\Article\Articles\Art\Art.\Section\Sections\Sec\Sec.\Title\"
        SectionLine$ = SectionLine$ + "Titles\Paragraph\Subparagraph\Chapter\Chapters\"
END IF
'************************** S T A R T ***************************************
CLS
FOR DocNum& = Beg TO Fin
  FreSp& = FRE("")
  QPrintRC "Document" + STR$(DocNum&), 15, 25, -1
  AbstrLine$ = ""
  FGetRT AbstrNdxFile, AbstrNdx, DocNum&, LenNdx
  NewSentFlag = TRUE
  NumberOfLines& = AbstrNdx.Last - AbstrNdx.First + 1
  IF NumberOfLines& > 1 AND NumberOfLines& < 300 THEN
          REDIM HighLights$(1 TO NumberOfLines&) 'highlighting information, just
          i = 1                                  'temporary keep it.
          FOR LineNum& = AbstrNdx.First TO AbstrNdx.Last
                  FGetRT AbstrFile, Txt, LineNum&, LenTxt
```

```
                AbstrLine$ = AbstrLine$ + Txt.Str + " "
                Highlight$(i) = Txt.Rest
                i = i + 1
PRINT Txt.Str
 NEXT
 Start = 1
 DO
        CombFlag = FALSE: SingFlag = FALSE: SectionFlag = FALSE: HyphenFlag = FALSE
        UpCaseLet = InstrTbl(Start, AbstrLine$, UpTable$)
        IF UpCaseLet > 0 THEN          'is there upper-case word?
                IF UpCaseLet <= 2 THEN     ' the first word in the abstract?
                        NewSentFlag = TRUE
                ELSE
                  'check if it is preceeded by '.' or ":" or ")" , i.e. it is a new sentence
                        PrevPos = 2
                        ch$ = MID$(AbstrLine$, UpCaseLet - 2, 1)
                        DO WHILE (ch$ < "a" OR ch$ > "z") AND (ch$ < "A" OR ch$ > "Z") AND InstrTbl2(1, ch$, ".:)") = 0 AND Up
                                PrevPos = PrevPos + 1
                                ch$ = MID$(AbstrLine$, UpCaseLet - PrevPos, 1)
                        LOOP
                        IF InstrTbl2(1, ch$, ".:)") THEN
                                NewSentFlag = TRUE
                        ELSE
                                NewSentFlag = FALSE
                        END IF
                END IF
'extract the upper-case word
LastLet = INSTR(UpCaseLet, AbstrLine$, " ")
FirstWord$ = MID$(AbstrLine$, UpCaseLet, LastLet - UpCaseLet)
'take out commas,quotes etc.
ch$ = RIGHT$(FirstWord$, 1)
DO WHILE (ch$ < "a" OR ch$ > "z") AND (ch$ < "A" OR ch$ > "Z")
        IF LEN(FirstWord$) > 1 THEN
                FirstWord$ = LEFT$(FirstWord$, LEN(FirstWord$) - 1)
                ch$ = RIGHT$(FirstWord$, 1)
        ELSE
                EXIT DO
        END IF
LOOP
'if it is UU-ll don't sanitize it
HyphenLoc = INSTR(FirstWord$, "-")
IF HyphenLoc THEN
        IF MID$(FirstWord$, HyphenLoc + 1, 1) >= "a" AND MID$(FirstWord$, HyphenLoc + 1, 1) <= "z" THEN
                HyphenFlag = TRUE
        END IF
END IF
IF NOT HyphenFlag THEN
        'compare word with noise word list if it a new sentence only
        IF NewSentFlag THEN
                NumNoise = UBOUND(Noise$)
                CALL FindExact(BYVAL VARPTR(Noise$(1)), NumNoise, FirstWord$)
        ELSE
                NumNoise = -1
        END IF
        IF NumNoise = -1 THEN  'it is not a noise word, check for the
                REDIM WordPhrase$(1 TO 5)                       'combined kw
                WordPhrase$(1) = LCASE$(FirstWord$)
                WPos = NotInstr(LastLet + 1, AbstrLine$, NoFirst$)
                IF WPos > 0 THEN
                        i = 1
                        DO
                          LastLet = INSTR(WPos, AbstrLine$, " ")
                          NextWord$ = MID$(AbstrLine$, WPos, LastLet - WPos)
                          ch$ = RIGHT$(NextWord$, 1)
                          DO WHILE (ch$ < "a" OR ch$ > "z") AND (ch$ < "A" OR ch$ > "Z")
                                IF LEN(NextWord$) > 1 THEN
                                        NextWord$ = LEFT$(NextWord$, LEN(NextWord$) - 1)
                                        ch$ = RIGHT$(NextWord$, 1)
                                ELSE
                                        EXIT DO
                                END IF
                          LOOP
                          IF LEN(NextWord$) > 1 THEN
                                WordPhrase$(i + 1) = LCASE$(NextWord$)
                                i = i + 1
                          END IF
                          WPos = NotInstr(LastLet + 1, AbstrLine$, NoFirst$)
                        LOOP UNTIL i = 4 OR WPos = 0
                        FindCombKey WordPhrase$(), CombKeyWordEMSX, CombFlag
                        IF NOT CombFlag THEN 'not a combinec kw check further
                                IF NOT NewSentFlag THEN
                                        'extract previous word
                                        Prev = QInstrB(UpCaseLet - PrevPos, AbstrLine$, " ")
                                        PrevWord$ = MID$(AbstrLine$, Prev + 1, UpCaseLet - 2 - Prev)
                                        'take out commas,quotes etc.
                                        ch$ = LEFT$(PrevWord$, 1)
                                        DO WHILE (ch$ < "a" OR ch$ > "z") AND (ch$ < "A" OR ch$ > "Z")
                                                IF LEN(PrevWord$) > 1 THEN
                                                        PrevWord$ = MID$(PrevWord$, 2)
                                                        ch$ = LEFT$(PrevWord$, 1)
                                                ELSE
                                                        EXIT DO
                                                END IF
                                        LOOP
                                        'check for comb. kw, beginning from previous word
                                        FOR i = 1 TO 4
                                                WordPhrase$(i + 1) = WordPhrase$(1)
                                        NEXT
                                        WordPhrase$(1) = LCASE$(PrevWord$)
                                        FindCombKey WordPhrase$(), CombKeyWordEMSX, CombFlag
                                END IF  'not NewSentFlag
                        END IF     'NOT CombFlag with previous word
```

```
END IF 'WPos -the last word in the abstract
IF NOT CombFlag THEN 'still not a combined kw check further
        'check against "Section", etc.
        IF INSTR(SectionLine$, "\" + FirstWord$ + "\") THEN
        SectionFlag = TRUE
END IF
IF NOT SectionFlag THEN
        'check for ending of the word against not-name ending
        NotNameEndFlag = FALSE
        IF Lang$ = "GERMAN" THEN
                FOR i = 3 TO 9
                        IF INSTR(GermanNotName$(i), "\" + RIGHT$(FirstWord$, i) + "\")
                                NotNameEndFlag = TRUE
                                FindSingKey FirstWord$, SingKeywordEMSZ, NewSentFlag,
                                EXIT FOR
                        END IF
                NEXT
        ELSE 'english
                FOR i = 2 TO 6
                        IF INSTR(EnglishNotName$(i), "\" + RIGHT$(FirstWord$, i) + "\"
                                NotNameEndFlag = TRUE
                                FindSingKey FirstWord$, SingKeywordEMSZ, NewSentFlag,
                                EXIT FOR
                        END IF
                NEXT
        END IF
        IF NOT NotNameEndFlag THEN
          'ending of the word was not found, check further
          'check if the previous word is an article
                IF NOT NewSentFlag THEN
                        Prev = QInstrB(UpCaseLet - PrevPos, AbstrLine$, " ")
                        PrevWord$ = MID$(AbstrLine$, Prev + 1, UpCaseLet - 2 - Prev)
                        PrevWord$ = "\" + PrevWord$ + "\"
                        IF INSTR(Articles1$, LCASE$(PrevWord$)) THEN
                                FindSingKey FirstWord$, SingKeywordEMSZ, NewSentFlag,
                        END IF
                END IF
                IF NOT SingFlag THEN
                        'no articles1 check further
                        'check for ending of the word against name ending
                        NameEndFlag = FALSE
                        IF Lang$ = "GERMAN" THEN
                                FOR i = 1 TO 2
                                        IF INSTR(GermanName$(i), "\" + RIGHT$(FirstWo
                                                NameEndFlag = TRUE
                                                FindSingKey FirstWord$, SingKeywordEM
                                                EXIT FOR
                                        END IF
                                NEXT
                        ELSE
                                FOR i = 1 TO 4
                                        IF INSTR(EnglishName$(i), "\" + RIGHT$(FirstW
                                                NameEndFlag = TRUE
                                                FindSingKey FirstWord$, SingKeywordEM
                                                EXIT FOR
                                        END IF
                                NEXT
                        END IF
                        IF NOT NameEndFlag THEN
                         'ending of the word was not found, check further
                                'check if the previous word is an article2
                                IF NOT NewSentFlag THEN
                                        IF INSTR(Articles2$, LCASE$(PrevWord$)) THEN
                                                IF Lang$ = "GERMAN" THEN
                                                        FindSingKey FirstWord$, SingKe
                                                ELSE
                                                        FindSingKey FirstWord$, SingKe
                                                END IF.
                                        END IF
                                END IF
                                IF NOT SingFlag THEN
                                        'no articles2 check further
                                        IF LEN(FirstWord$) >= 8 THEN    'no sense to
                                                FindSingKey FirstWord$, SingKeywordEM
                                        END IF
                                        IF NOT SingFlag THEN GOSUB ChangeWord
                                END IF
                        ELSE 'if NameEndFlag = TRUE
                                IF NOT SingFlag THEN GOSUB ChangeWord
                        END IF 'NameEndFlag
                END IF    ' SingFlag
        ELSE        'if NotNameEndFlag = TRUE
                IF NOT SingFlag THEN GOSUB ChangeWord
        END IF        'NotNameEndFlag
                        END IF          'SectionFlag
                END IF              'NOT CombFlag without previous word
        END IF                  'not a noise word
    END IF                  'not HyphenFlag
  END IF                'UpCaseLet > 0
  IF CombFlag THEN
        Start = UpCaseLet + NextStart
  ELSE
        Start = UpCaseLet + LEN(FirstWord$)
  END IF
LOOP UNTIL UpCaseLet = 0
REDIM Text$(1 TO 1)
CALL Wrapping(AbstrLine$, Text$(), NumLines)
FOR i = 1 TO NumberOfLines&
    IF INSTR(Text$(i), "XX") THEN  'this line was changed, save it
        Txt.Str = Text$(i)
        Txt.Rest = Highlight$(i)
```

```
'        PRINT #1, Txt.Str
                FPutRT AbstrFile, Txt, CLNG(AbstrNdx.First + i - 1), LenTxt
              END IF
          NEXT
  END IF 'NumberOfLines& valid range
  q$ = INKEY$
  IF q$ = CHR$(27) THEN EXIT FOR
NEXT 'document
FClose AbstrFile
END

ChangeWord:
        'not a single kw - change it!
        IF RIGHT$(FirstWord$, 1) = "/" THEN FirstWord$ = LEFT$(FirstWord$, LEN(FirstWord$) - 1)
        MID$(AbstrLine$, UpCaseLet, LEN(FirstWord$)) = STRING$(LEN(FirstWord$), 88)
RETURN

SUB Config STATIC

Cmd$ = COMMAND$
Parms = InCount(Cmd$, " ") + 1   '-- number of parameters
IF Parms = 4 THEN

        '-- Expected information on command line:
        '   Config file, First Doc, Last Doc

        Extract Cmd$, " ", 1, Strt, SLen '-- extract first parm
        DBName$ = MID$(Cmd$, Strt, SLen)
        ConfigFile$ = DBName$ + ".CFG"

        Extract Cmd$, " ", 2, Strt, SLen '-- extract second parm
        Machine$ = MID$(Cmd$, Strt, SLen)

        Extract Cmd$, " ", 3, Strt, SLen '-- extract third parm
        Beg = VAL(MID$(Cmd$, Strt, SLen))

        Extract Cmd$, " ", 4, Strt, SLen '-- extract fourth parm
        Fin = VAL(MID$(Cmd$, Strt, SLen))



ELSE

        PRINT
        PRINT "SANITIZE Program Error: Missing Parameters"
        PRINT
        PRINT
        PRINT "Required Parameters are:"
        PRINT
        PRINT "SANITIZE  Config File  Machine  First Doc  Last Doc"
        PRINT
        Chime 10
        PRINT "Press the SPACE BAR to exit:"
        i$ = INPUT$(1)
        END

END IF

OPEN ConfigFile$ FOR INPUT ACCESS READ SHARED AS #1
  INPUT #1, Fg, Bg, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$
CLOSE #1

i = 1
OPEN LstDir$ + "NOISE.DAT" FOR INPUT ACCESS READ SHARED AS #1
DO UNTIL EOF(1)
  REDIM PRESERVE Noise$(1 TO i)
  INPUT #1, Noise$(i)
  n$ = LEFT$(Noise$(i), LEN(Noise$(i)) - 1)
  Noise$(i) = UCASE$(LEFT$(n$, 1)) + MID$(n$, 2)
  i = i + 1
LOOP
CLOSE #1

IF NOT EmsLoaded% THEN
        Chime 8
        PRINT "The EMS has not been loaded."
        STOP
END IF


SixteenK = 16 * 1024
SixtyFour = 64
ThirtyTwo = 32
ThirtyTwoK& = ThirtyTwo * 1024&


END SUB

SUB EmsAlloc (NumPages%, Handle%, LoadFILE$) STATIC

EmsAllocMem NumPages%, Handle%
IF EmsError% THEN
        PRINT "Couldn't allocate"; CLNG(NumPages) * SixteenK; "bytes of EMS for "; LocFILE$
        Chime 2
        DO: LOOP UNTIL LEN(INKEY$) = 0
        i$ = INPUT$(1)
        END
END IF
```

```
END SUB

SUB FindCombKey (WordPhrase$(), KeyEMS%, CombFlag) STATIC

CombFlag = FALSE
DIM KeyTemp AS CombKeyType '— entire Combined Keyword
LENKey = LEN(KeyTemp)
Slash$ = "///"
ASCslash = ASC("/")
        '— if it's a valid range, then check words in range
        IF FirstLast%(WordPhrase$(1), First, Last, Comb) THEN
                FOR j = Last TO First STEP -1
                '— get word from Combined Keyword List (COMBKEY.STR)
                EasGetTEl KeyTemp, LENKey, j, KeyEMS%
                '— convert it to a variable-length string for speed
                KeyTemp$tr$ = RTRIM$(KeyTemp.Str)
                Words = InCount(KeyTempStr$, " ") + 1'count number of words
                IF Words <= 4 THEN
                        CALL Extract(KeyTempStr$, " ", 1, Strt, SLen)'extract first word
                        CurrKey$ = MID$(KeyTempStr$, Strt, SLen)'     of combined keyword

                        IF MidChar%(CurrKey$, SLen) = ASCslash THEN
                                Exact = TRUE
                                CurrKey$ = LEFT$(CurrKey$, SLen - 1)
                                SLen = SLen - 1
                        ELSE
                                Exact = FALSE
                        END IF
                        'compare first word of combined key [CurrKey$]
                        'against the current document word [WordTempStr$]

                        IF Lang$ = "GERMAN" THEN
                                IF NOT Exact THEN
                                        Match = (CurrKey$ = LEFT$(WordTempStr$, SLen))
                                ELSE ' check for *exact* match
                                        Match = (CurrKey$ = WordTempStr$)
                                END IF
                        ELSE
                                IF NOT Exact THEN
                                        Match = (LCASE$(CurrKey$) = LEFT$(WordPhrase$(1), SLen))
                                ELSE ' check for *exact* match
                                        Match = (LCASE$(CurrKey$) = WordPhrase$(1))
                                END IF
END IF

' no match, skip to next combined key in the First-Last range
IF NOT Match GOTO SkipCombKey

' continue matching the rest of the words in the combined key
' exiting out as soon as there's a non-match

AtFlag = FALSE
NotFlag = FALSE

FOR k = 2 TO Words' number of words left in combined key

        ' extract the next word from the current combined keyword (j)
        CALL Extract(KeyTempStr$, " ", k, Strt, SLen)
        CurrKey$ = MID$(KeyTempStr$, Strt, SLen)

        IF MidChar%(CurrKey$, SLen) = ASCslash THEN
                Exact = TRUE
                CurrKey$ = LEFT$(CurrKey$, SLen - 1)
                SLen = SLen - 1
        ELSE
                Exact = FALSE
        END IF

        IF AtFlag = FALSE AND NotFlag = FALSE THEN
                DocWord$ = WordPhrase$(k)
        ELSE
                IF AtFlag = FALSE AND NotFlag = TRUE THEN
                        DocWord$ = WordPhrase$(k + 1)
                ELSE
                        IF AtFlag = TRUE AND NotFlag = FALSE THEN
                                DocWord$ = WordPhrase$(k - 1)
                        ELSE
                                DocWord$ = WordPhrase$(k)
                        END IF
                END IF
        END IF

        IF English THEN Lower DocWord$

        IF ASCII(CurrKey$) <> ASC("@") THEN
                IF Lang$ = "GERMAN" THEN
                '— German: no need to use Lcase$
                        IF Exact THEN ' check for *exact* match
                                Match = (CurrKey$ = DocWord$)
                        ELSE ' wildcard match, only compare # of chars in CurrKey$
                                Match = (CurrKey$ = LEFT$(DocWord$, SLen))
                        END IF
                ELSE
                        IF Exact THEN ' check for *exact* match
                                Match = (LCASE$(CurrKey$) = DocWord$)
                        ELSE ' wildcard match, only compare # of chars in CurrKey$
                                Match = (LCASE$(CurrKey$) = LEFT$(DocWord$, SLen))
                        END IF
                END IF
        ELSE '— special processing for @ wildcard
                IF INSTR(AtList$, "/" + DocWord$ + "/") THEN
                        Match = TRUE' the word was in the @ list, so continue
```

```
                ELSE
                        IF English THEN
                                Match = FALSE
                        ELSE
                                Match = TRUE
                                AtFlag = TRUE
                        END IF
                END IF
                IF Match THEN
                        DocWord$ = WordPhrase$(k + 1)
                        IF DocWord$ = "not" OR DocWord$ = "be" OR DocWord$ = "nicht" THEN NotFlag = TRUE
                END IF
        END IF

        IF NOT Match GOTO SkipCombKey

NEXT' word in current combined keyword

IF Match THEN ' this is a combined keyword, so add it to the list
        CombFlag = TRUE

                                NextStart = LEN(KeyTempStr$)
                                EXIT FOR
                        END IF
                END IF
SkipCombKey:
                NEXT
        END IF' Table range was valid

END SUB

SUB FindSingKey (Word$, KeyEMS%, NewSentFlagFS, SingFlag, Limit) STATIC

Count = 0
IF LEN(Word$) < Limit THEN EXIT SUB  'it can't satisfied us
DIM KeyTemp AS SingKeyType '-- Single Keyword to be compared
LENKey = LEN(KeyTemp)
ASCslash = ASC("/")
SingFlag = FALSE
IF RIGHT$(FirstWord$, 2) = "'s" THEN FirstWord$ = LEFT$(FirstWord$, LEN(FirstWord$) - 2)
IF LEN(Word$) < 3 THEN Word$ = Word$ + STRING$(3 - LEN(Word$), ASC("/"))
IF Lang$ = "GERMAN" THEN Lower Word$
' check if the first 3 letters of the word return
' a valid range from the 3-dimensional table array
  IF FirstLast%(LCASE$(Word$), First, Last, Sing) THEN  ' yes, so search thru range
          DO
                FOR j = Last TO First STEP -1

                        '-- get the word from the SINGKEY.STR list
                                EmsGet1El KeyTemp, LENKey, j, KeyEMS%
                                CurrKey$ = RTRIM$(KeyTemp.Str)
                                SLen = LEN(CurrKey$)

                        '-- compare the single keyword [CurrKey$/KeyTemp.Str]
                        '   against the document word [Word$]
                        '== [replaced] IF RIGHT$(CurrKey$, 1) = "/" THEN

                        IF MidChar%(CurrKey$, SLen) = ASCslash THEN
                                CurrKey$ = LEFT$(CurrKey$, SLen - 1)
                                Match = (CurrKey$ = Word$)
                        ELSE
                                Match = (CurrKey$ = LEFT$(Word$, SLen))
                        END IF

                        IF Match THEN
                                IF LEN(CurrKey$) >= Limit THEN
                                        SingFlag = TRUE
                                        EXIT FOR
                                END IF
                        END IF

                NEXT' key in range
        IF NewSentFlagFS THEN
                        'now check against  upper- and low-case words, even if they are in the middle
                        IF Count = 0 THEN  'try low case
                                Word$ = LCASE$(Word$)
                        END IF
                Count = Count + 1
        END IF
          LOOP UNTIL Count > 1 OR SingFlag
        END IF' the range was valid

END SUB

DEFSNG A-Z
'*************************************************
FUNCTION FirstLast% (Word$, First%, Last%, KeyType%) STATIC

'-- returns the starting (First) and ending (Last) range for the word
'   by looking it up in the Table%() array
First% = 0
a0 = ASCII(Word$)
IF a0 > 37 AND a0 < 123 THEN
        a1 = MidChar(Word$, 2)
        IF a1 > 37 AND a1 < 123 THEN
                a2 = MidChar(Word$, 3)
                IF a2 > 37 AND a2 < 123 THEN
                        a = XLateTable%(a0)
                        b = XLateTable%(a1)
                        c = XLateTable%(a2)
                        IF a = 0 OR b = 0 OR c = 0 THEN FirstLast% = 0: EXIT FUNCTION
```

```
                    IF KeyType% = Sing THEN
                        First% = SingTable%(a, b, c, 1)
                        Last% = SingTable%(a, b, c, 2)
                    ELSE
                        First% = CombTable%(a, b, c, 1)
                        Last% = CombTable%(a, b, c, 2)
                    END IF
                END IF
            END IF
        END IF
    END IF
    '-- Return FALSE if there was no valid range (i.e., First%=0)
    FirstLast% = (First% <> 0)

    END FUNCTION

    DEFINT A-Z
    SUB LoadData STATIC

    '------------------------------------------------ Read in Combined Keys

    LoadFILE$ = LstDir$ + "COMBKEY.STR"
    IF NOT Exist%(LoadFILE$) THEN CLS : PRINT LoadFILE$; " not found.": END
    DIM CombKeyTemp AS STR$64
    NumCombKeyword = FileSize&(LoadFILE$) \ SixtyFour
    CombKeyWordEMS% = LoadIntoEMS%(LoadFILE$)


    '------------------------------------------------ Read Single Keys

    LoadFILE$ = LstDir$ + "SINGKEY.STR"
    IF NOT Exist%(LoadFILE$) THEN CLS : PRINT LoadFILE$; " not found.": END
    DIM SingKeyTemp AS STR$32
    NumKeyword = FileSize&(LoadFILE$) \ ThirtyTwo


    SingKeywordEMS% = LoadIntoEMS%(LoadFILE$)


    '------------------------------------------------ Read 3-Char Tables

    Symb = 28: First = 1: Last = 2
    REDIM XLateTable%(38 TO 122)
    REDIM SingTable%(1 TO Symb, 1 TO Symb, 1 TO Symb, 1 TO 2)
    REDIM CombTable%(1 TO Symb, 1 TO Symb, 1 TO Symb, 1 TO 2)

    XLateTable(47) = 1' / char, as used in non-wildcard words
    XLateTable(38) = 2' & char, as used in S&P, A&P, etc.
    ASCa = ASC("a")
    ASCz = ASC("z")
    FOR i = ASCa TO ASCz
        XLateTable(i) = i - 94' so that a=3, b=4,...,z=28
    NEXT

    FGetAH LstDir$ + "KEYWORD.TBL", SEG SingTable%(1, 1, 1, 1), (4 * 28), (28 * 28)
    FGetAH LstDir$ + "KEYCOMB.TBL", SEG CombTable%(1, 1, 1, 1), (4 * 28), (28 * 28)

    END SUB

    FUNCTION LoadIntoEMS (File$) STATIC

    '-------- Returns the handle where the file was loaded into --------

    EMSPg = EmsGetPFSeg%
    SizeofFile& = FileSize&(File$)

    NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
    EmsAlloc NumPages, FileEMS, File$

    Num32kBlocks = SizeofFile& \ ThirtyTwoK&
    LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
    FOpenAll File$, 0, 4, LoadFILE
    FOR i = 1 TO Num32kBlocks + 1

        Box0 14, 10, 18, 70, 2, -1
        PaintBox0 14, 10, 18, 70, -1
        QPrintRC "Loading " + File$ + " block" + STR$(i) + " /" + STR$(Num32kBlocks - 1) + " ", 16, 12, -1

    '-- map pages of the EMS memory to the EMS upper mem page frame
    FOR j = 1 TO 2
        EmsMapMem FileEMS, j, (i - 1) * 2 + j
        IF EmsError% THEN PRINT "Ems error:"; EmsError%: STOP
    NEXT

    '-- seek to beginning of current block
    FSeek LoadFILE, (i - 1) * ThirtyTwoK&

    IF DOSError% THEN PRINT "Dos Error:"; WhichError%: STOP

    IF i < Num32kBlocks + 1 THEN

        '-- get the 32k block and put it directly into the EMS page frame
        FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, ThirtyTwoK&

        IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

    ELSE

        '-- Load the left over (<32k) bytes
        FGetA LoadFILE, BYVAL EMSPg, BYVAL 0, LeftOver&
```

```
        IF DOSError% THEN PRINT "Dos Error:"; ErrorMsg$(WhichError%): STOP

    END IF
NEXT

FClose LoadFILE

ClearScr0 14, 10, 18, 70, NormAttr

LoadIntoEM$ = FileEMS

END FUNCTION

SUB Wrapping (Sent$, Text$(), NumLines) STATIC
Wid = 78
MaxNumLines = LEN(Sent$) \ Wid + 3
REDIM Text$(1 TO MaxNumLines)
NumLines = 0
DO
    '-- increment NumLinese counter for number of NumLineses of text
    NumLines = NumLines + 1
    IF NumLines > MaxNumLines THEN
      REDIM PRESERVE Text$(1 TO NumLines)
    END IF
    '-- look for the last space so we can word wrap at that point
    NewSent = QInstrB%(Wid + 1, Sent$, CHR$(1))
    IF NewSent > 2 THEN
        IF RIGHT$(RTRIM$(Sent$), 1) = CHR$(1) THEN
            Text$(NumLines) = RTRIM$(LEFT$(Sent$, NewSent))
        ELSE
            Text$(NumLines) = RTRIM$(LEFT$(Sent$, NewSent - 1))
        END IF
        Sent$ = MID$(Sent$, NewSent + 1)
    ELSE
        LastSpc = QInstrBX(Wid + 1, Sent$, " ")
        Text$(NumLines) = RTRIM$(LEFT$(Sent$, LastSpc))
        '-- remove portion of string that's been moved to the Text$() array
        Sent$ = MID$(Sent$, LastSpc + 1)
    END IF
LOOP WHILE LEN(Sent$) > Wid
Sent$ = RTRIM$(Sent$)
IF LEN(Sent$) THEN
    NumLines = NumLines + 1
    IF NumLines > MaxNumLines THEN
      REDIM PRESERVE Text$(1 TO NumLines)
    END IF
    Text$(NumLines) = Sent$
END IF
END SUB


DEFINT A-Z
'$TITLE: 'Q-Search User Program'
'$SUBTITLE: 'QSEARCH Module'

CONST FALSE = 0, TRUE = NOT FALSE, ASCEND = 0, DESCEND = 1
CONST MaxShow = 50

'scan code + 200 for not to mix with letters
CONST UP = 272, PgUp = 273, Dn = 280, PgDn = 281, HM = 271, EN = 279
CONST CtrlPgUp = 332, CtrlPgDn = 318, CtrlHM = 319, CtrlEN = 317
CONST F1 = 259, F2 = 260, F3 = 261, F4 = 262, F5 = 263
CONST F6 = 264, F7 = 265, F8 = 266, F9 = 267, F10 = 268

CONST ESC = 27, CR = 13
CONST NewSearch = 1, AddWords = 2, EditSearch = 3, Back = 4, Forward = 5, SWAP$ = 6, EX = 7

'$INCLUDE: '\\VADIM\C-DRIVE\user\INCLUDE\TYPES.BI'
'$INCLUDE: '\\VADIM\C-DRIVE\user\include\defcnf.bi'
'$INCLUDE: '\\VADIM\C-DRIVE\user\INCLUDE\QEdiType.BI'
'$INCLUDE: '\\VADIM\C-DRIVE\user\INCLUDE\shared.BI'

'---------------------------------------------------- External Declarations
'$INCLUDE: '\\VADIM\C-DRIVE\user\EXTERN.BAS'

DECLARE SUB InitMem (Segm%, Addr%, NumBytes%, Value%)
DECLARE SUB QEdit (Array$(), x%, Action%, Ed AS EditInfo)
DECLARE SUB FindExact (BYVAL Address%, NumEl%, Search1$)
'---------------------------------------------------- Internal Declarations
DECLARE SUB AddSearchTerms (Expr AS ExpressionType)
DECLARE SUB AddSentence (Expr AS ExpressionType)
DECLARE SUB BuildCombTable (Mode$)
DECLARE SUB ChangeChar (Txt$, New$, Keep$)
DECLARE SUB ClearBG ()
DECLARE SUB Code2Str (Store$, Location%, Code%)
DECLARE SUB Config ()
DECLARE SUB CPrint (x$)
DECLARE SUB CreateTables ()
DECLARE SUB DeleteWord (Expr AS ExpressionType)
DECLARE SUB CutWord (Kword$)
DECLARE SUB DispMsg (Msg$, RX, CX)
DECLARE SUB DispOSL (LibName$, ScrnName$)
DECLARE SUB DrawBox (ULRow%, ULCol%, LRRow%, LRCol%, Frame%, Col%)
DECLARE SUB EmsAlloc (NumPages%, Handle%, LoadFile$)
DECLARE SUB FindCombKey (WordList$(), NumWord%, CombFound$(), NumCombFound%)
DECLARE SUB FindSingKey (WordList$(), NumWord%, SingFound$(), NumSingFound%)
DECLARE SUB Freehandles ()
DECLARE SUB FullText (First&, Last&, FileNum%, Expr$)
DECLARE SUB HistMessage ()
DECLARE SUB InsertWord (Expr AS ExpressionType)
DECLARE SUB Lib2Scrn (NameInLib$, ScrnLib%(), MonoCode%, Attribute%, ErrorCode%)
```

```
DECLARE SUB LoadData ()
DECLARE SUB LoadPrefixes (Prefixes$(), MeanPrefixes$(), Lang$)
DECLARE SUB OtherWords (Expr AS ExpressionType, WordPtr)
DECLARE SUB PickChoice (a() AS ANY, NumX, Label$, PickX(), NumPickX, ExactFlag%, Expr AS ExpressionType)
DECLARE SUB PrintAbstr (First&, Last&, FileNum, RecNum&)
DECLARE SUB RankRecords (Recs() AS RecInfoType, NumFoundX, E AS ExpressionType)
DECLARE SUB ReadEnglishText (Txt$)
DECLARE SUB ReadGermanText (Txt$)
DECLARE SUB References (Text$)
DECLARE SUB ReWriteHist (CurrX)
DECLARE SUB Sections (Txt$)
DECLARE SUB SelectMenu (Ch$, Expression AS ANY, HotStr$, GlobalStatus%)
DECLARE SUB ScrollHist (CurrX, Direction$, MaxNum)
DECLARE SUB ScrSR (SR$)
DECLARE SUB ShowAbstr (RecNum&)
DECLARE SUB ShowDoc (Show() AS RecInfoType, NumShow, Expr AS ExpressionType, RecNum)
DECLARE SUB ShowExpr (Expr AS ExpressionType)
DECLARE SUB ShowHist (NumShow, RecNumX, Expr AS ExpressionType)
DECLARE SUB ShowKeywords (RecNum&)
DECLARE SUB ShowQuery ()
DECLARE SUB TermMatch (Term$, Expr AS ExpressionType, a() AS CollectType, NumX%, Repeats, ExactFlag)
DECLARE SUB VertMenu (Item$(), ChoiceX, MaxLenX, BoxBotX, Ky$, ActionX, Cnf AS Config, Mode$, HX, EK, PgUX, PgDX, UX, DX, TermTypeMode
DECLARE SUB WaitSpace ()
DECLARE SUB WindMgr (ULRowX, ULColX, LRRowX, LRColX, FrameX, BoxColrX, TextColrX, Text$)
DECLARE SUB WordParse (Txt$, WordList$(), NumWordsX)

DECLARE FUNCTION BoxInput$ (Edit$, Title$, Prompt$, RowX, ColX, Scan)
DECLARE FUNCTION ComboSum& (Bit$, Value&, Mode$, Poly!())
DECLARE FUNCTION Dict$ (CodeX)
DECLARE FUNCTION DictSrchX (w AS DictType)
DECLARE FUNCTION FirstLastX (Word$, FirstX, LastX, KeyTypeX)
DECLARE FUNCTION KeyInstr% (KeyStr$, Srch$)
DECLARE FUNCTION KeyMid$ (KeyStr$, StartX)
DECLARE FUNCTION Question$ (Prompt$, Choice$, Label$)
DECLARE FUNCTION Num$ (xX)
DECLARE FUNCTION SpaceNum$ (xX, SpaceX)
DECLARE FUNCTION Str2Code% (SS, kX)
DECLARE FUNCTION ZeroNum$ (xX, ZeroX)

'-----------------------------------------------------------------------

ON ERROR GOTO Again

STACK 6000
ReStart:
REDIM Menu$(1)
REDIM GermanMenu$(1)
LoadedFlag = FALSE
CALL Config
DIM Expression AS ExpressionType
Exclude$ = ""

MenuData:
DATA "Menu Selections:"
DATA "Related Topics      | T"
DATA "Add Search Words    | A"
DATA "Delete a Word       | D"
DATA "Restore Deleted Word | R"
DATA "-------------------"
DATA "View Documents      | V"
DATA "-------------------"
DATA "New Search          | N"
DATA "-------------------"
DATA "Exit Program        | X"
DATA "END"

MenuGermanData:
DATA "MENUE:"
DATA "Verwandte Themen    | T"
DATA "Fuege Wort Hinzu    | F"
DATA "Loesche Wort        | L"
DATA "Hol Wort Zurueck    | H"
DATA "-------------------"
DATA "Suche Fundstelle    | S"
DATA "-------------------"
DATA "Neue Sucheingabe    | N"
DATA "-------------------"
DATA "Exit System         | X"
DATA "END"

NLGData:
DATA "mr","mrs","messrs","sen","rep","ms","dr","drs"
SectionData:
DATA "Section","Sec","Sec."
ArticleData:
DATA "Article","Art.","Art"
ParagraphData:
DATA "Paragraph","Par","Par."
ArtikelData:
DATA "Artikel","Art","Art."
NumData:
DATA "I","II","III","IV","V","VI","VII","VIII","IX","X"
DATA "XI","XII","XIII","XIV","XV","XVI","XVII","XVIII","XIX","XX"
DATA "XXI","XXII","XXIII","XXIV","XXV","XXVI","XXVII","XXVIII","XXIX","XXX"


EnglishStr$ = "TADRVNX"
GermanStr$ = "TFLHSNX"
IF Lang$ = "GERMAN" THEN
       HotStr$ = GermanStr$
```

```
ELSE
        HotStr$ = EnglishStr$
END IF

'$Page
'_____________________________________________________________
'                    START OF PROGRAM FLOW
'_____________________________________________________________

NewSearch:

        ClearBG
        QPrintRC "| Q-Search |", 1, 36, RevAttr%
        GlobalStatus = NewSearch
        Expression.Num = 0      ' Number of subexpressions entered (none yet!)
        Expression.Match = -1   ' No full search yet
        Expression.SubExpr(1).Match = -1    ' No search yet
        Expression.SubExpr(1).Num = 0       ' No matches either
        Expression.SubExpr(1).Phrase = ""

        CurrentSub = 1 ' current subexpr (1 to 3) that's being 'edited'
        MaxWordLen = 0 ' reset max word length global variable for ShowExpr
        WordPtr = 0    ' clear the WordPtr for FindRelatives()
        LIFO$ = ""
        OriginalExpr$ = ""

'---- Get the sentence input and place words directly into Expression

AddSentence Expression
ShowExpr Expression

'_____________________ SEARCH MENU _____________________

Menu:

IF Lang$ <> "GERMAN" THEN
        CALL References("PRESS LETTER OR USE " + CHR$(24) + CHR$(25) + " TO HIGHLIGHT CHOICE AND THEN PRESS ENTER")
ELSE
        CALL References("SHOULD BE MESSAGE IN GERMAN")
END IF

Action = 1' set initial action to just display menu
Ky$ = "Search MENU"
Choice = 0

LOCATE 2, 51
IF Lang$ = "GERMAN" THEN
        VertMenu GermanMenu$(), Choice, LEN(GermanMenu$(1)), 18, Ky$, Action, Cnf, "M", 0, 0, 0, 0, 0, 0, TermTypeMode$, TermTypeFlag
ELSE
        VertMenu Menu$(), Choice, LEN(Menu$(1)), 18, Ky$, Action, Cnf, "M", 0, 0, 0, 0, 0, 0, TermTypeMode$, TermTypeFlag
END IF

'---- Action is automatically set to 3 for polling after call to VertMenu

Choice = 1
DO
        IF Lang$ = "GERMAN" THEN
                VertMenu GermanMenu$(), Choice, LEN(Menu$(1)), 18, Ky$, Action, Cnf, "M", 0, 0, 0, 0, 0, 0, TermTypeMode$, TermTypeFla
        ELSE
                VertMenu Menu$(), Choice, LEN(Menu$(1)), 18, Ky$, Action, Cnf, "M", 0, 0, 0, 0, 0, 0, TermTypeMode$, TermTypeFlag
        END IF

        IF Action = 3 THEN 'user exited without making a choice or ESCaping

                IF LEN(Ky$) = 1 THEN ' check for direct access (single letter) Choice

                        IF Choice THEN EXIT DO
                        LETTER = INSTR(MenuChoice$, Ky$)
                        IF LETTER THEN
                                Choice = LETTER
                                EXIT DO
                        END IF
                IF LEN(Ky$) = 1 THEN
                        AscKy = ASC(Ky$)
                ELSEIF LEN(Ky$) = 2 THEN
                        AscKy = ASC(RIGHT$(Ky$, 1)) + 200
                END IF
                        SELECT CASE AscKy

                                CASE F10, NewSearchKey' new search
                                        IF Lang$ = "GERMAN" THEN
                                                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                                                IF i$ = "J" THEN
                                                        GOTO NewSearch
                                                END IF
                                        ELSE
                                                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                                                IF i$ = "Y" THEN
                                                        GOTO NewSearch
                                                END IF
                                        END IF

                                CASE F1' help
                                        CALL DispMsg("Sorry, there is currently no Help available.  Press the Space Bar to Return to M
                                        CALL WaitSpace
                                        CALL DispMsg("", 0, 0)

                                CASE F2, ShowExprKey 'Show query
                                        CALL ShowQuery

                                CASE ELSE
```

```
                END SELECT
            END IF

        ELSEIF Action = 4 THEN ' user exited with a choice, but check for ESCape

            IF Ky$ = CHR$(ESC) THEN
                GOTO ExitSearch
            ELSE
                EXIT DO
            END IF

        ELSE
            LOCATE 1, 25
            Chime 2
            PRINT " -- Invalid ACTION value "; Action; " returned -- "
            Freehandles
            WaitSpace
            END
        END IF
    LOOP

    Action = 5' restore screen
    VertMenu Menu$(), Choice, LEN(Menu$(1)), 18, Ky$, Action, Cnf, "N", 0, 0, 0, 0, 0, 0, TermTypeModes, TermTypeFlag
    LOCATE , , 0

    ScrSR "S"
    IF Choice = 0 OR (Expression.Num = 0 AND Choice <> 8 AND Choice <> 10) GOTO Menu
    CALL SelectMenu(MID$(MenuChoice$, Choice, 1), Expression, HotStr$, GlobalStatus)
    '=============================
    IF GlobalStatus <> EX THEN 'not exit
        ScrSR "R"
        ShowExpr Expression
        IF GlobalStatus = NewSearch GOTO NewSearch
        GOTO Menu
        '=============================
    ELSE 'exit
    ExitSearch:
        IF Lang$ = "GERMAN" THEN
            i$ = Question$("Exit Program? [J/N]", "NJ", "ACHTUNG!")
        ELSE
            i$ = Question$("Exit Program? [Y/N]", "NY", "WARNING!")
        END IF
        IF i$ = "N" THEN
            ShowExpr Expression
            GOTO Menu
        END IF
    ExitProgram:
        CLS
        Freehandles
        END
        DATA "Copyright 1990, 1991 by Ted M. Young. All Rights Reserved."
    END IF
    '$Page
    Again:
    NormAttr = OneColor%(Fg, BG)
    'PRINT ERR
    CALL ClearBG
    DispMsg "Sorry, cannot continue with this action.  Please try another.", 0, 0
    DO: LOOP UNTIL LEN(INKEY$)
    IF LoadedFlag THEN
        RESUME NewSearch
    END IF
    END

    SUB AddSentence (Expr AS ExpressionType) STATIC

    '-- let's the user type in a full sentence, parses the sentence and
    '   places the keywords found into the current expression

    DIM Ed AS EditInfo

    NumExprWords = 0

    GetSentence:

    Ed.Rows = 3
    Ed.Wide = 76
    Ed.Wrap = Ed.Wide
    Ed.AColor = NormAttr
    Ed.Frame = 0
    Ed.CurCol = 1: Ed.LC = 1
    Ed.CurLine = 1: Ed.TL = 1

    REDIM Sentence$(1 TO 3) '-- up to 3 lines of text

    UlRow = 5: UlCol = 2
    LRRow = UlRow + Ed.Rows + 1: LRCol = UlCol + Ed.Wide + 1

    ScrSR "S"
    ClearScr0 UlRow, UlCol, LRRow, LRCol, NormAttr
    CALL DrawBox(UlRow, UlCol, LRRow, LRCol, 1, NormAttr)
    IF TermTypeFlag THEN
        LMChar$ = "├"
        RMChar$ = "┤"
    ELSE
        LMChar$ = "["
        RMChar$ = "]"
    END IF
    IF Lang$ = "GERMAN" THEN
        Title$ = "Stellen Sie Ihre Frage"
```

```
ELSE
        Title$ = "Enter Your Search Request"
END IF
CALL QPrintRC(LMChar$ + Title$ + RMChar$, ULRow, (ULCol + LRCol - LEN(Title$) - 2) \ 2, NormAttr)

IF Lang$ = "GERMAN" THEN
        Title$ = "danach ENTER Taste"
ELSE
        Title$ = "Press ENTER when done"
END IF
CALL QPrintRC(LMChar$ + Title$ + RMChar$, LRRow, (ULCol + LRCol - LEN(Title$) - 2) \ 2, NormAttr)

LOCATE ULRow + 1, ULCol + 1

x$ = ""
QEdit Sentence$(), x$, 0, Ed

IF ASC(x$) = ESC THEN EXIT SUB

ScrSR "R"

'-- put the sentence into a single string
SearchReq$ = ""
FOR i = 1 TO 3
        IF LEN(Sentence$(i)) THEN SearchReq$ = SearchReq$ + " " + Sentence$(i)
NEXT
SearchReq$ = LTRIM$(SearchReq$)

IF LEN(SearchReq$) = 0 GOTO GetSentence

IF Lang$ = "GERMAN" THEN
        DispMsg "Computer liest Sucheingabe", R, c
ELSE
        DispMsg "Parsing sentence...", R, c
END IF
IF Lang$ = "ENGLISH" THEN
        ReadEnglishText SearchReq$
ELSEIF Lang$ = "GERMAN" THEN
        ReadGermanText SearchReq$
END IF
REDIM WordList$(1 TO 1)
WordParse SearchReq$, WordList$(), NumWords

REDIM Comb$(1 TO 1)
FindCombKey WordList$(), NumWords, Comb$(), NumComb

REDIM Sing$(1 TO 1)
FindSingKey WordList$(), NumWords, Sing$(), NumSing

'---- Now use DictSrch% to convert the synonyms found to their Code Numbers
DIM DictTemp AS DictType
REDIM ExprCodes(1 TO 1) AS CodePolyType

FOR i = 1 TO NumComb
        DictTemp.Str = Comb$(i)
        GOSUB AddWord
NEXT

FOR i = 1 TO NumSing
        DictTemp.Str = Sing$(i)
        GOSUB AddWord
NEXT
IF NumSing + NumComb = 0 OR Expr.Num = 0 THEN
        DispMsg "", 0, 0
        IF NumSing + NumComb = 0 THEN
                IF Lang$ = "GERMAN" THEN
                        DispMsg "Sucheingabe enthaelt kein bekanntes Wort. LEERTASTE um weiterzumachen!", R, c
                ELSE
                        DispMsg "There were no keywords found in your sentence.  Press the Space Bar now to continue.", R, c
                END IF
        ELSE
                IF Lang$ = "GERMAN" THEN
                        DispMsg "Kein Dokument enthaelt diesen Begriff.", R, c
                ELSE
                        DispMsg "That word/phrase is not an indexed word in this database.  Press the Space Bar now to continue.", R,
                END IF
        END IF
        WaitSpace
        DispMsg "", 0, 0
        GOTO GetSentence
END IF

SortT ExprCodes(1), NumExprWords, DESCEND, LEN(ExprCodes(1)), 2, -3

Expr.Num = 1
Expr.SubExpr(1).Num = NumExprWords

FOR k = 1 TO NumExprWords
        Code2Str Expr.SubExpr(1).Phrase, k, ExprCodes(k).Code
        OriginalExpr$ = OriginalExpr$ + MKI$(ExprCodes(k).Code)
NEXT

DispMsg "", 0, 0
EXIT SUB

'-----------------------------------------

AddWord:

LETTER$ = LEFT$(DictTemp.Str, 2)
```

```
IF LETTER$ = "zs" OR LETTER$ = "za" THEN
        SecNum = VAL(MID$(DictTemp.Str, 3))
        IF SecNum > 0 AND SecNum <= 3000 THEN
                IF LETTER$ = "zs" THEN
                        Code = SecNum + SecCode
                ELSE
                        IF SecNum <= 30 THEN Code = SecNum + ArtCode
                END IF
        END IF
ELSE
        Code = DictSrchZ(DictTemp)
END IF

IF Code THEN   '— then it's a valid code

        Found = FALSE
        FOR j = 1 TO NumExprWords
                IF Code = ExprCodes(j).Code THEN '— this code was already entered
                        Found = TRUE
                        EXIT FOR
                END IF
        NEXT

        IF NOT Found THEN '— add it to the expression
                IF NumExprWords < 15 AND KYIndx(Code).Num > 0 THEN
                        .Expr.Num = 1
                        NumExprWords = NumExprWords + 1
                        REDIM PRESERVE ExprCodes(1 TO NumExprWords) AS CodePolyType
                        ExprCodes(NumExprWords).Code = Code
                        EasGet1El ExprCodes(NumExprWords).Poly, LEN(ExprCodes(1).Poly), Code, PolySemyEMS

                END IF

        END IF

ELSE            '— it's not a valid code, so indicate an error

        Chime 2
        DispMsg "The word " + RTRIM$(DictTemp.Str) + " was not found in the dictionary.", R, c
        WaitSpace
        DispMsg "", 0, 0
        Freehandles
        END

END IF

RETURN

END SUB

SUB ChangeChar (Txt$, New$, Keep$)

'*************************************************************
'can be deleted after change abstract program
Start = 1
DO
ApostrLoc = INSTR(Start, Txt$, " '")
IF ApostrLoc THEN
        Start = ApostrLoc + 1
        SpaceLoc = INSTR(Start, Txt$, " ")
        IF SpaceLoc > 0 THEN
                MID$(Txt$, ApostrLoc, SpaceLoc - ApostrLoc + 1) = STRING$(SpaceLoc - ApostrLoc + 1, 32)
                Start = ApostrLoc
        ELSE
                EXIT DO
        END IF
END IF
LOOP UNTIL NOT ApostrLoc

'the same for the first word
ApostrLoc = INSTR(Start, Txt$, "'")
IF ApostrLoc = 1 THEN
        SpaceLoc = INSTR(Start, Txt$, " ")
        IF SpaceLoc > 0 THEN
                MID$(Txt$, ApostrLoc, SpaceLoc - ApostrLoc + 1) = STRING$(SpaceLoc - ApostrLoc + 1, 32)
                Start = ApostrLoc
        END IF
END IF
'delete "' "
Start = 1
DO
ApostrLoc = INSTR(Start, Txt$, "' ")
IF ApostrLoc THEN
        Start = ApostrLoc + 1
        MID$(Txt$, ApostrLoc, 1) = " "
END IF
LOOP UNTIL NOT ApostrLoc

''*************************************************************

'— replace all chars except contained in Keep$
LENTxt = LEN(Txt$)
FOR i = 1 TO LENTxt
        IF INSTR(Keep$, MID$(Txt$, i, 1)) = 0 THEN
                MID$(Txt$, i, 1) = New$
        END IF
NEXT

Start = InstrTbl(1, Txt$, Keep$)
IF Start > 0 THEN
```

```
MID$(Txt$, 1, Start - 1) = STRING$(Start - 1, 1)
        DO
                SpaceLoc = INSTR(Start, Txt$, " ")
                IF SpaceLoc THEN
                        DO WHILE MidChar(Txt$, SpaceLoc + 1) = 32
                                MID$(Txt$, SpaceLoc, 1) = CHR$(0)
                                SpaceLoc = SpaceLoc + 1
                        LOOP
                        Start = SpaceLoc
                ELSE
                        EXIT DO
                END IF
        LOOP UNTIL SpaceLoc = 0
END IF

END SUB

SUB ClearBG STATIC

STATIC Array%, BGX() '-- FALSE if this is the first call, so we need to save
                                                    '    the screen in an array after we fill the background
                                                    '    TRUE if we've already stored the bg in the array
IF TermTypeFlag THEN
        IF Array% THEN

                ScrnRest0 1, 1, 25, 80, BGX(1)

        ELSE

                CLS
                Ruler1$ = STRING$(1, 178) + STRING$(78, 178) + STRING$(1, 178)
                Ruler2$ = STRING$(1, 178) + STRING$(78, 176) + STRING$(1, 178)
                Ruler3$ = STRING$(1, 178) + STRING$(78, 178) + STRING$(1, 178)

                QPrintRC Ruler1$, 1, 1, NormAttr%
                FOR i = 2 TO 24
                        QPrintRC Ruler2$, i, 1, NormAttr%
                NEXT
                QPrintRC Ruler3$, 25, 1, NormAttr%

                REDIM BGX(1 TO 2000)
                ScrnSave 1, 1, 25, 80, BGX(1)

                Array = TRUE

        END IF
ELSE
        CLS
END IF
END SUB

SUB Code2Str (Store$, Location%, Code%) STATIC

MID$(Store$, Location% * 2 - 1, 2) = MKI$(Code%)

END SUB

SUB Config STATIC

SHARED NLG$(), LENNLG(), Menu$(), MenuChoice$, GermanMenu$(), GermanMenuChoice$
CLS
IF COMMAND$ <> "" AND COMMAND$ <> "/B" THEN
        IF INSTR(COMMAND$, " ") THEN
                DBName$ = LEFT$(COMMAND$, INSTR(COMMAND$, " ") - 1)
        ELSE
                DBName$ = COMMAND$
        END IF
ELSE
        PRINT "No Database Specified.  Please type: USER [database name]"
        Chime 10
        END
END IF

OPEN DBName$ + ".CFG" FOR INPUT AS #1
  INPUT #1, Fg, BG, Brdr, LstDir$, DocDir$, NdxDir$, AbstrDir$, Lang$, TermTypeMode$
CLOSE #1

Cnf.MonTyp = Monitor%
IF INSTR(COMMAND$, "/B") OR TermTypeMode$ <> "LOCAL" THEN Cnf.MonTyp = 2

SELECT CASE Cnf.MonTyp
        CASE 3, 5, 7                    'CGA, EGA/Color or VGA/Color monitors
                Cnf.PulBar = 48
                Cnf.MenBox = 49
                Cnf.ActivCh = 48
                Cnf.InActCh = 52'6
                Cnf.HiLite = 31'79
                Cnf.InActHiLt = 64
                Cnf.NonMen = 30
                Cnf.CurSize = 13
                IF Cnf.MonTyp = 5 THEN Cnf.CurSize = 7
        CASE ELSE
                Cnf.PulBar = 112
                Cnf.MenBox = 112'2
                Cnf.ActivCh = 112'10
                Cnf.InActCh = 112'4
                Cnf.HiLite = 15'112
                Cnf.InActHiLt = 80
                Cnf.NonMen = 7
                Cnf.CurSize = 13
```

```
END SELECT

File$ = NdxDir$ + "AVGDOCFQ.DAT"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

OPEN File$ FOR INPUT AS #1
        INPUT #1, RootofAvgDocFreq!
CLOSE #1

File$ = NdxDir$ + "PolyAvg.Dat"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

OPEN File$ FOR INPUT AS #1
        INPUT #1, PolyAvg!
CLOSE #1

File$ = LstDir$ + "A" + Lang$ + ".LST"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

OPEN File$ FOR INPUT AS #1
        LINE INPUT #1, A$List$
CLOSE #1

'COLOR Fg, BG, Brdr

'vvvvvvvvvvvvvvvvvvvvvvvvv
'which terminal time
IF TermTypeMode$ = "LOCAL" OR TermTypeMode$ = "PC" THEN
        TermTypeFlag = TRUE
ELSE
        TermTypeFlag = FALSE
END IF
IF TermTypeFlag THEN
        HistChar$ = CHR$(178)
        HighChar$ = CHR$(219)
        LMarg$ = "▌"
        RMarg$ = "▐"
ELSE
        HistChar$ = "X"
        HighChar$ = "*"
        LMarg$ = "["
        RMarg$ = "]"
END IF
'---------------------------
NormAttr = OneColorX(Fg, BG)
IF Cnf.MonTyp = 2 THEN
        IF BG > 0 THEN
                RevAttr = OneColorX(14, 4)
        ELSE
                RevAttr = OneColorX(BG, Fg AND 7)
        END IF
ELSE
        IF BG > 0 THEN
                RevAttr = OneColorX(14, 4)
                HiAttr = OneColor(0, 3)
        ELSE
                RevAttr = OneColorX(BG, Fg AND 7)
                HiAttr = RevAttr
        END IF
END IF

CLS

IF NOT EmsLoaded THEN
        PRINT "The EMS Driver has not been loaded."
        Chime 2
        END
END IF

HomeKey = ASC("H")
EndKey = ASC("E")
PgUpKey = ASC("P")
PgDnKey = ASC("G")
RightArrowKey = ASC("R")
LeftArrowKey = ASC("L")
UpArrowKey = ASC("U")
DownArrowKey = ASC("D")
ShowExprKey = ASC("S")
DirNumKey = ASC("I")
NewSearchKey = ASC("N")

SixtyFour = 64
SixteenK = 16384
ThirtyTwoK$ = 32768
Sing = 0: Comb = 1
```

```
AscA = ASC("a")
ASCZ = ASC("z")
AscUpperA = ASC("A")
ASCUpperZ = ASC("Z")
ASC0 = ASC("0")
ASC9 = ASC("9")
Boost! = 2!
AbstrMode = 8
New$ = " "      ' replacement string for punctuation
Keep$ = ""
FOR j = 65 TO 90: Keep$ = Keep$ + CHR$(j): NEXT
FOR j = 97 TO 122: Keep$ = Keep$ + CHR$(j): NEXT
Keep$ = Keep$ + "!&"                                    '+"([-" + CHR$(34)

NdxLEN = LEN(Ndx)
FCInfoLEN = LEN(FCInfo)
KYInfoLEN = LEN(KYInfo)
WeightLEN = LEN(Weight)
PolyLEN = LEN(PolyValue)

CLS

' Display the opening title screen -- it's in a separate QSCR library
IF TermTypeFlag THEN
        DispQSL "USER.QSL", "OPENING"
ELSE
  DispQSL "USER.QSL", "OPENTERM"
END IF
CreateTables
IF Lang$ = "GERMAN" THEN
        QPrintRC " [ EINLADEN DER DATEN ] ", 24, 29, NormAttr
ELSE
        QPrintRC " [ LOADING DATA FILES ] ", 24, 29, NormAttr
END IF
LoadData

Chime 6

NumFilesLoaded = NumFilesLoaded + 1

IF Lang$ = "GERMAN" THEN
        QPrintRC " [ EINLADEN DER DATEN " + STR$(NumFilesLoaded) + " ] ", 24, 29, NormAttr
ELSE
        QPrintRC " [ LOADING DATA FILES " + STR$(NumFilesLoaded) + " ] ", 24, 29, NormAttr
END IF

IF Lang$ <> "GERMAN" THEN
        REDIM Menu$(0 TO 0)
        RESTORE MenuData
        m = 0
        DO
                READ x$
                IF x$ = "END" THEN EXIT DO
                REDIM PRESERVE Menu$(0 TO m)
                IF NOT TermTypeFlag THEN
                        a = INSTR(x$, "|")
                        IF a THEN MID$(x$, a, 1) = "|"
                        IF LEFT$(x$, 1) = "-" THEN x$ = STRING$(24, "-")
                END IF
                Menu$(m) = x$
                m = m + 1
        LOOP
ELSE
        REDIM GermanMenu$(0 TO 0)
        RESTORE MenuGermanData
        m = 0
        DO
                READ x$
                IF x$ = "END" THEN EXIT DO
                REDIM PRESERVE GermanMenu$(0 TO m)
                IF NOT TermTypeFlag THEN
                        a = INSTR(x$, "|")
                        IF a THEN MID$(x$, a, 1) = "|"
                        IF LEFT$(x$, 1) = "-" THEN x$ = STRING$(24, "-")
                END IF
                GermanMenu$(m) = x$
                m = m + 1
        LOOP
END IF

'create menuchoice$ (a list of valid keys to pick from menu)
'extract the characters from the menu display itself since the keys
'are always listed as the last character on the line

MenuChoice$ = SPACE$(m - 1)
IF Lang$ = "GERMAN" THEN
        FOR i = 1 TO m - 1
                x$ = RIGHT$(GermanMenu$(i), 1)
                '-- make the "-" choices into chr(0) so they can't be picked
                IF x$ = "-" THEN x$ = CHR$(0)
                MID$(MenuChoice$, i, 1) = x$
        NEXT
ELSE
        FOR i = 1 TO m - 1
                x$ = RIGHT$(Menu$(i), 1)
                '-- make the "-" choices into chr(0) so they can't be picked
                IF x$ = "-" THEN x$ = CHR$(0)
                MID$(MenuChoice$, i, 1) = x$
        NEXT
END IF
```

```
IF Lang$ <> "GERMAN" THEN

        REDIM NLGS(1 TO 8), LENNLG(1 TO 8)
        RESTORE NLGData
        FOR i = 1 TO 8
                READ NLGS(i)
                LENNLG(i) = LEN(NLGS(1))
        NEXT

        REDIM Section$(1 TO 3)
        RESTORE SectionData
        FOR i = 1 TO 3
                READ Section$(i)
        NEXT
        REDIM Article$(1 TO 3)
        RESTORE ArticleData
        FOR i = 1 TO 3
                READ Article$(i)
        NEXT

ELSE

        REDIM Paragraph$(1 TO 3)
        RESTORE ParagraphData
        FOR i = 1 TO 3
                READ Paragraph$(i)
        NEXT
        REDIM Artikel$(1 TO 3)
        RESTORE ArtikelData
        FOR i = 1 TO 3
                READ Artikel$(i)
        NEXT

END IF

REDIM Numbers$(1 TO 30)
RESTORE NumData
FOR i = 1 TO 30
        READ Numbers$(i)
NEXT

CALL LoadPrefixes(Prefixes$(), MeanPrefixes$(), Lang$)

END SUB

SUB CPrint (x$) STATIC

END SUB

SUB CreateTables STATIC

'— Load in the 3-Letter index tables from disk into EMS

Symb = 28
First = 1: Last = 2
REDIM XLateTableX(38 TO 122)
NumEl = Symb ^ 3

XLateTableX(47) = 1' / char, as used in non-wildcard words
XLateTableX(38) = 2' & char, as used in S&P, A&P, etc.
FOR i = ASC("a") TO ASC("z")
        XLateTableX(i) = i - 96' so that a=3, b=4,...,z=28
NEXT

REDIM SingTableFirX(1 TO NumEl)
FGetAH LstDir$ + "KEYWORD.FIR", SEG SingTableFirX(1), 2, NumEl
Array2EMS SingTableFirX(1), 2, NumEl, SingTblFirEMS
ERASE SingTableFirX

REDIM SingTableLasX(1 TO NumEl)
FGetAH LstDir$ + "KEYWORD.LAS", SEG SingTableLasX(1), 2, NumEl
Array2EMS SingTableLasX(1), 2, NumEl, SingTblLasEMS
ERASE SingTableLasX

REDIM CombTableFirX(1 TO NumEl)
FGetAH LstDir$ + "KEYCOMB.FIR", SEG CombTableFirX(1), 2, NumEl
Array2EMS CombTableFirX(1), 2, NumEl, CombTblFirEMS
ERASE CombTableFirX

REDIM CombTableLasX(1 TO NumEl)
FGetAH LstDir$ + "KEYCOMB.LAS", SEG CombTableLasX(1), 2, NumEl
Array2EMS CombTableLasX(1), 2, NumEl, CombTblLasEMS
ERASE CombTableLasX

END SUB

SUB CutWord (KWord$)

'— cut KWord$ by "-" or by " " or just truncates more than 24 letters

IF LEN(KWord$) > 24 THEN
        KWord$ = LEFT$(KWord$, 21) + "..."
        '— we'd prefer to cut off the word at a "-"
END IF

END SUB

SUB Define (Word$) STATIC

REDIM ScrX(800)' 5 rows of 80 columns, including window
```

```
MScrnSave 21, 1, 25, 80, ScrX(1)
ClearScr0 21, 1, 25, 80, RevAttr
CALL DrawBox(21, 1, 25, 80, 1, RevAttr)

IF Lang$ = "GERMAN" THEN
        QPrintRC "Definition von '" + Word$ + "':", 23, 4, -1
ELSE
        QPrintRC "Definition of '" + Word$ + "':", 23, 4, -1
END IF

WaitSpace

MScrnRest 21, 1, 25, 80, SEG ScrX(1)
ERASE ScrX

END SUB

FUNCTION Dict$ (Code%) STATIC

'== This function translates a code# to the actual word/phrase

IF Code < 0 THEN ' ---- highlight this word because it's a SWAP$ relative
        Code = ABS(Code)
        Flag$ = CHR$(250)
ELSE
        Flag$ = " "
END IF

'---- get the "head" word for this code
DIM DictTemp AS DictType
CALL EmsGet1El(SEG DictTemp, LEN(DictTemp), CodeX, DictWordH%)

IF Code > 0 THEN

        '---- return the frequency of the word prefixed to the word itself

        'Dict$ = SpaceNum$(Code, 4) + " " + SpaceNum$(KYIndx(Code).Num, 4) + Flag$ + RTRIM$(DictTemp.Str)

        EmsGet PolyValue, PolyLEN, CLNG(Code), PolySemyEMS
'   Dict$ = SpaceNum$(KYIndx(Code).Num, 5) + " " + RTRIM$(DictTemp.Str)

'*  use the following line instead to display poly value
  Dict$ = LEFT$(LTRIM$(STR$(PolyValue.Value)) + "     ", 5) + " " + SpaceNum$(KYIndx(Code).Num, 5) + Flag$ + RTRIM$(DictTemp.Str)

ELSE ' this word wasn't found?

        Dict$ = " Cbla-<> "

END IF

END FUNCTION

FUNCTION DictSrch% (Word AS DictType) STATIC

'== Binary searches thru an array of DictType (word & code#) for a
'== word and returns the code #.  The EMS Handle and the number of
'== DictCode entries are Common Shared.

DIM DictTemp AS DictType, DictTemp2 AS DictType
LENDict = LEN(DictTemp)

L = 1: R = DictCodeNum% ' total number of code entries

DO

        x = (CLNG(L) + R) \ 2
        EmsGet1El DictTemp, LENDict, x, DictCodeH%

        IF LCASE$(Word.Str) < LCASE$(DictTemp.Str) THEN
                R = x - 1
        ELSE
                IF LCASE$(Word.Str) <> LCASE$(DictTemp.Str) OR Lang$ <> "ENGLISH" THEN
                        L = x + 1
                ELSE

                        '-- save it temporarily in case we need to restore it
                        '   if the forward/backward look doesn't bring any
                        '   positive results (i.e., we didn't get a match, and we
                        '   need the original DictTemp for the LOOP test)

                        SWAP DictTemp2, DictTemp
                        EmsGet1El DictTemp, LENDict, x + 1, DictCodeH%
                        IF Word.Str = DictTemp.Str THEN
                                x = x + 1
                                EXIT DO
                        ELSE
                                EmsGet1El DictTemp, LENDict, x - 1, DictCodeH%
                                IF Word.Str = DictTemp.Str THEN
                                        x = x + 1

                                        EXIT DO
                                END IF
                        END IF
                        SWAP DictTemp, DictTemp2
                        L = L + 1
                END IF
        END IF
LOOP UNTIL Word.Str = DictTemp.Str OR L > R
```

```
IF Word.Str = DictTemp.Str THEN '-- it was found, so return where

        DictSrch% = DictTemp.Code
        Word.Code = DictTemp.Code

ELSE '-- wasn't found, return 0

        DictSrch% = 0
        Word.Code = 0

END IF

END FUNCTION

SUB DispMsg (Msg$, R%, c%) STATIC

STATIC WindOpen, Scr%() ' is there already a message displayed?

LOCATE , , 0 ' turn the cursor off

'---- if the window is open, see if this is a "close window" message
IF WindOpen THEN
        IF Msg$ = "" THEN
                '---- Msg$ is null, so close the window and exit this sub
                GOSUB MsgClose
                EXIT SUB
        ELSE
                '---- Msg$ wasn't null, but the window is open, so it's an error
                CALL Chime(9)
                GOSUB MsgClose
        END IF
END IF

Wid = LEN(Msg$)
IF Wid > 60 THEN Wid = 60

Msg$ = Msg$ + " " ' make sure there's a space to find at the end (see below)

MaxLin = LEN(Msg$) \ Wid + 3
IF MaxLin > 23 THEN MaxLin = 23
REDIM Text$(MaxLin)
Lin = 0

DO

        '-- increment line counter for number of lines of text
        Lin = Lin + 1

        '-- look for the last space so we can word wrap at that point
        LastSpc = QInstrB(Wid + 1, Msg$, " ")
        Text$(Lin) = RTRIM$(LEFT$(Msg$, LastSpc))

        '-- remove portion of string that's been moved to the Text$() array
        Msg$ = MID$(Msg$, LastSpc + 1)

LOOP WHILE LEN(Msg$) > Wid

Msg$ = RTRIM$(Msg$)
IF LEN(Msg$) THEN
        Lin = Lin + 1
        Text$(Lin) = Msg$
END IF

'---- recompute width based on word-wrapped text
Wid = 0
FOR i = 1 TO Lin
        IF LEN(Text$(i)) > Wid THEN Wid = LEN(Text$(i))
NEXT

IF R <> 0 AND c = 0 THEN
        ULr = R
ELSE
        ULr = 11
END IF
  DULr = ULr - 1
  LRr = ULr + Lin + 1
  DLRr = LRr + 2

horizmargin = (80 - Wid) \ 2
  ULc = horizmargin
  DULc = ULc - 3
  LRc = 80 - INT(horizmargin)
  IF Wid / 2 = Wid \ 2 THEN LRc = LRc + 1
  DLRc = LRc + 1

REDIM Scr%(ArraySize%(DULr, DULc, DLRr, DLRc))
ScrnSave0 DULr, DULc, DLRr, DLRc, SEG Scr%(0)
WindMgr ULr, ULc, LRr, LRc, 4, NormAttr, NormAttr, ""
FOR i = 1 TO Lin
        QPrintRC Text$(i), ULr + i, ULc + 1, -1
NEXT

R = ULr + Lin
c = ULc + 1 + LEN(Text$(Lin))
IF LEN(Text$(Lin)) + 2 = Wid THEN c = ULc + 1: R = R + 1

ERASE Text$
WindOpen = TRUE

EXIT SUB

'------------------------------------------------------------ close window
```

```
MsgClose:

ScrnRestO DULr, DULc, DLRr, DLRc, SEG ScrZ(0)
ERASE ScrX
WindOpen = FALSE
RETURN

END SUB

SUB EmsAlloc (NumPages%, HandleX, LoadFile$) STATIC

CALL EmsAllocMem(NumPages%, HandleX)
IF EmsErrorZ THEN
        PRINT "Couldn't allocate"; CLNG(NumPages) * SixteenK / 1024; "Kbytes of EMS for "; LoadFile$
        CALL Freehandles
        CALL Chime(2)
        END
END IF

END SUB

SUB FindCombKey (WordList$(), NumWord%, CombFound$(), NumCombFoundX) STATIC

'ARRAY NAME   LEN DESCRIPTION              DIRECTION  MODIFIED?
'------------ --- ---------------------- ---------- ----------
'CombKeyFILE   64 Combined Keyword List   (Shared)   (Unchanged)
'CombFound$() VAR Combined Keywords Found (Returned) (Changed)
'WordList$()  VAR Document Words          (Passed)   (Unchanged)

DIM KeyTemp AS CombKeyType' for interface to EmsGet1El
LENKeyTemp = LEN(KeyTemp)

NumCombFound = 0

FOR i = 1 TO NumWord - 1' number of words to process
        AtFlag = FALSE
        NotFlag = FALSE
        '---- make lower case since combined keywords ignore case
        Word$ = LCASE$(WordList$(i))

        IF Word$ = "" GOTO SkipCombKey

        '---- if it's a valid range, then do comparisons for words in the range
        IF FirstLastX(Word$, First, Last, Comb) THEN

             FOR j = Last TO First STEP -1

                  FGetRT CombKeyFILE, KeyTemp, CLNG(j), LENKeyTemp
Words = InCount(RTRIM$(KeyTemp.Str), " ") + 1'count number of words

'---- if the keyword has more words than are left in the word list
'     skip it, because there's no possibility of a match.
IF Words > NumWord - i + 1 THEN
        IF Lang$ = "GERMAN" THEN      'AtFlag should work only for German
             IF INSTR(KeyTemp.Str, "@") = 0 GOTO SkipCombKey
        ELSE
             GOTO SkipCombKey
        END IF
END IF
CALL Extract(KeyTemp.Str, " ", 1, Strt, Slen)'extract first word
CurrKey$ = MID$(KeyTemp.Str, Strt, Slen)'     of combined keyword

IF RIGHT$(CurrKey$, 1) = "/" THEN
        Exact = TRUE
        CurrKey$ = LEFT$(CurrKey$, Slen - 1)
        Slen = Slen - 1
ELSE
        Exact = FALSE
END IF

'---- compare first word of combined key [CurrKey$]
'     against the current document word [Word$]

IF RIGHT$(Word$, 1) = "/" THEN Word$ = LEFT$(Word$, LEN(Word$) - 1)

IF Exact THEN ' check for "exact" match
        Match = (LCASE$(CurrKey$) = LCASE$(Word$))
ELSE
        Match = (LCASE$(CurrKey$) = LCASE$(LEFT$(Word$, Slen)))
END IF

' no match, skip to next combined key in the First-Last range
IF NOT Match GOTO SkipCombKey

' continue matching the rest of the words in the combined key
' exiting out as soon as there's a non-match

FOR k = 1 TO Words - 1' number of words left in combined key

        ' extract the next word from the current combined keyword (j)
        CALL Extract(KeyTemp.Str, " ", k + 1, Strt, Slen)
        CurrKey$ = MID$(KeyTemp.Str, Strt, Slen)

        IF RIGHT$(CurrKey$, 1) = "/" THEN
             Exact = TRUE
             CurrKey$ = LEFT$(CurrKey$, Slen - 1)
             Slen = Slen - 1
        ELSE
             Exact = FALSE
        END IF

        '---- Find AB(German only) AäB, & AänotB in Doc if AäB is in Dict
```

```
IF AtFlag = FALSE AND NotFlag = FALSE AND i + k <= NumWord THEN
        DocWord$ = WordList$(i + k)
ELSE
        IF AtFlag = FALSE AND NotFlag = TRUE AND i + k < NumWord THEN
                DocWord$ = WordList$(i + k + 1)
        ELSE
                IF AtFlag = TRUE AND NotFlag = FALSE AND k > 1 THEN
                        DocWord$ = WordList$(i + k - 1)
                ELSE
                        DocWord$ = WordList$(i + k)
                END IF
        END IF
END IF

IF RIGHT$(DocWord$, 1) = "/" THEN DocWord$ = LEFT$(DocWord$, LEN(DocWord$) - 1)

IF CurrKey$ = "@" THEN ' special processing for @ wildcard
        IF INSTR(AtList$, "/" + DocWord$ + "/") THEN
                Match = TRUE' the word was in the @ list, so continue
        ELSE
                IF Lang$ = "ENGLISH" THEN
                        Match = FALSE
                ELSE
                        Match = TRUE
                        AtFlag = TRUE

                END IF
        END IF
ELSE
        IF Exact THEN ' check for "exact" match
                Match = (LCASE$(CurrKey$) = LCASE$(DocWord$))
        ELSE ' wildcard match, only compare # of chars in CurrKey$
                Match = (LCASE$(CurrKey$) = LCASE$(LEFT$(DocWord$, Slen)))
        END IF
END IF'
        IF Match AND i + k < NumWord THEN
                DocWord$ = WordList$(i + k + 1)
                IF DocWord$ = "not" OR DocWord$ = "be" OR DocWord$ = "nicht" THEN NotFlag = TRUE
        END IF

        IF NOT Match THEN EXIT FOR

NEXT' word in current combined keyword

IF Match THEN ' this is a combined keyword, so add it to the list
        NumCombFound = NumCombFound + 1
        REDIM PRESERVE CombFound$(1 TO NumCombFound)
        CombFound$(NumCombFound) = RTRIM$(KeyTemp.Str)

        '-- mark combined word so that single keys
        '   are not generated from parts of combined keys found
        IF NotFlag AND NOT AtFlag THEN          'if A @ not @ as A @ B
                FOR k = i TO i + Words
                        WordList$(k) = WordList$(k) + CHR$(255)
                NEXT
        ELSE
                IF NOT NotFlag AND AtFlag THEN          'if A@ as A @ B
                        FOR k = i TO i + Words - 2
                                WordList$(k) = WordList$(k) + CHR$(255)
                        NEXT
                ELSE                                    'normal procedure
                        FOR k = i TO i + Words - 1
                                WordList$(k) = WordList$(k) + CHR$(255)
                        NEXT
                END IF
        END IF
        EXIT FOR
    END IF
```

```
SkipCombKey:
            NEXT
      END IF' Table range was valid

NEXT' key in list

END SUB

SUB FindSingKey (WordList$(), NumWordX, SingFound$(), NumSingFoundX) STATIC

DIM KeyTemp AS SingKeyType' for interface to EmsGet1El
DIM ExclTemp AS DictType
LENKeyTemp = LEN(KeyTemp)
NumSingFound = 0
ExcludeAdd$ = ""
FOR i = 1 TO NumWord' number of words in document
        MeanPrefixFlag = FALSE: PrefixFlag = FALSE: CombineFlag = FALSE
        Word$ = WordList$(i)
        IF Word$ <> "" THEN
                IF RIGHT$(Word$, 1) = CHR$(255) THEN
                        Word$ = LEFT$(Word$, LEN(Word$) - 1)
                        CombineFlag = TRUE
                END IF
                IF Lang$ <> "GERMAN" THEN
                        SELECT CASE ASC(Word$)
                                CASE AscA TO AscZ: SearchStep = 1
                                CASE AscUpperA TO AscUpperZ:
                                        IF ASC(MID$(Word$, 2, 1)) < AscUpperA OR ASC(MID$(Word$, 2, 1)) > AscUpperZ T
                                                SearchStep = 2
                                        END IF
                                CASE ELSE: SearchStep = 3
```

```
                    END SELECT
                END IF
                ' check if the first 3 letters of the word return
                ' a valid range from the 3-dimensional table array
TryAgain:
.....
IF FirstLastX(LCASE$(Word$), First, Last, Sing) THEN ' yes, so search thru range
      FOR j = Last TO First STEP -1
            FGetRT SingKeyFILE, KeyTemp, CLNG(j), LENKeyTemp
            CurrKey$ = RTRIM$(KeyTemp.Str)
            Slen = LEN(CurrKey$)

            IF RIGHT$(CurrKey$, 1) = "/" THEN
                  Exact = TRUE
                  CurrKey$ = LEFT$(CurrKey$, Slen - 1)
                  Slen = Slen - 1
            ELSE
                  Exact = FALSE
            END IF

            'compare the single keyword [CurrKey$/KeyTemp.Str]
            'against the document word [Word$]
            IF Exact THEN ' check for "exact" match
                  Match = (CurrKey$ = Word$)
            ELSE ' check for wildcard match
                  Match = (CurrKey$ = LEFT$(Word$, Slen))
            END IF
            IF Match THEN
                  IF NOT CombineFlag THEN  'for both languages
                        ' add the single keyword to the list
                        NumSingFound = NumSingFound + 1
                        REDIM PRESERVE SingFound$(1 TO NumSingFound)
                        SingFound$(NumSingFound) = RTRIM$(KeyTemp.Str)
                        EXIT FOR
                  ELSE
                        'it is part of combin, so don't store it but add to the exclude list
                        ExclTemp.Str = RTRIM$(KeyTemp.Str)
                        Code = DictSrch(ExclTemp)
                        IF Code > 0 THEN ExcludeAdd$ = ExcludeAdd$ + MKI$(Code)
                        EXIT FOR
                  END IF
            END IF

      NEXT' key in range
      IF NOT Match AND Lang$ <> "GERMAN" THEN
            SELECT CASE SearchStep
                  CASE 1: Word$ = UCASE$(LEFT$(Word$, 1)) + LCASE$(RIGHT$(Word$, LEN(Word$) - 1))
                              SearchStep = SearchStep + 1
                              GOTO TryAgain
                  CASE 2: Word$ = UCASE$(Word$)
                              SearchStep = SearchStep + 1
                              GOTO TryAgain
                  CASE ELSE:
                              IF I = 1 THEN
                                    FirstLetter = ASC(LEFT$(LTRIM$(Word$), 1))
                                    IF FirstLetter >= 65 AND FirstLetter <= 90 THEN
                                          Word$ = LCASE$(Word$)
                                          GOTO TryAgain
                                    END IF
                              END IF
            END SELECT
      END IF   'not match
ELSE   'range was not valid
      LETTER$ = LEFT$(Word$, 2)
      IF LETTER$ = "zs" OR LETTER$ = "za" THEN
            'add the single keyword to the list
            NumSingFound = NumSingFound + 1
            REDIM PRESERVE SingFound$(1 TO NumSingFound)
            SingFound$(NumSingFound) = RTRIM$(Word$)
      END IF
END IF' the range was valid
```

```
' check for prefixes
IF NOT MeanPrefixFlag THEN
      'check for meaningful prefixes. If found, divide word in two parts
      Word$ = LCASE$(Word$)
      LenW = LEN(Word$)
      FOR NumLet = 14 TO 3 STEP -1
            IF LenW >= NumLet + 3 THEN  'should leave at least 3 letters
                  IF INSTR(MeanPrefixes$(NumLet), "\" + LEFT$(Word$, NumLet) + "\") THEN
                        WordTemp$ = MID$(Word$, NumLet + 1)
                        Word$ = LEFT$(Word$, NumLet)
                        MeanPrefixFlag = TRUE
                        EXIT FOR
                  END IF
            END IF
            NEXT
            IF MeanPrefixFlag THEN
                  IF Match THEN CombineFlag = TRUE
                  GOTO TryAgain 'check again
            END IF
      ELSE   'if MeanPrefixFlag
            IF WordTemp$ <> "" THEN  'CombineFlag still TRUE
                  Word$ = WordTemp$
                  WordTemp$ = ""
                  GOTO TryAgain
            END IF
      END IF  'MeanPrefixFlag
      'check for meaningless prefixes and delete it
      IF NOT PrefixFlag THEN  'only one time
```

```
                Word$ = LCASE$(Word$)
                LenW = LEN(Word$)
                FOR NumLet = 9 TO 2 STEP -1
                        IF LenW >= NumLet + 3 THEN  'should leave at least 3 letters
                                IF INSTR(Prefixes$(NumLet), "\" + LEFT$(Word$, NumLet) + "\") THEN
                                        Word$ = MID$(Word$, NumLet + 1)
                                        PrefixFlag = TRUE
                                        EXIT FOR
                                END IF
                        END IF
                NEXT
                IF PrefixFlag THEN
                      IF Match THEN CombineFlag = TRUE
                      GOTO TryAgain 'check again
                END IF
            END IF      'PrefixFlag
        END IF          'Word$ > " "
NEXT' word in document

END SUB

FUNCTION FirstLastX (Wordx$, FirstX, LastX, SingOrCombX) STATIC

' returns the starting (First) and ending (Last) range for the word
' by looking it up in the TableX() array

Word$ = RTRIM$(Wordx$)

IF LEN(Word$) < 3 THEN Word$ = Word$ + STRING$(3 - LEN(Word$), "/")

SpaceLoc = INSTR(Word$, " ")
IF SpaceLoc > 1 AND SpaceLoc < 4 THEN
        Word$ = LEFT$(Word$, SpaceLoc - 1) + STRING$(4 - SpaceLoc, "/")
END IF

a = XLateTableX(ASCII%(Word$))
b = XLateTableX(MidChar(Word$, 2))
c = XLateTableX(MidChar(Word$, 3))
IF a = 0 OR b = 0 OR c = 0 THEN FirstLast = 0: EXIT FUNCTION
Index& = (a - 1) * 784 + (b - 1) * 28 + c

IF SingOrCombX = Sing THEN
        TableFirEMS = SingTblFirEMS
        TableLasEMS = SingTblLasEMS
ELSE
        TableFirEMS = CombTblFirEMS
        TableLasEMS = CombTblLasEMS
END IF

EmsGet FirstX, 2, Index&, TableFirEMS
EmsGet LastX, 2, Index&, TableLasEMS

'Return FALSE if there was no valid range (i.e., FirstX=0)
FirstLastX = (FirstX <> 0)

END FUNCTION

SUB Freehandles STATIC

'---- Release EMS memory

IF DictCodeHX THEN EmsRelMem DictCodeHX
IF DictWordHX THEN EmsRelMem DictWordHX
IF SingTblFirEMS THEN EmsRelMem SingTblFirEMS
IF SingTblLasEMS THEN EmsRelMem SingTblLasEMS
IF CombTblFirEMS THEN EmsRelMem CombTblFirEMS
IF CombTblLasEMS THEN EmsRelMem CombTblLasEMS
IF Array1EMS THEN EmsRelMem Array1EMS
IF PolySemyEMS THEN EmsRelMem PolySemyEMS

'---- Close Files

IF KeyNdxFile THEN FClose KeyNdxFile
IF KYInvertDatFILE THEN FClose KYInvertDatFILE
IF FCInvertDatFILE THEN FClose FCInvertDatFILE
IF HeaderFILE THEN FClose HeaderFILE
IF SingKeyFILE THEN FClose SingKeyFILE
IF CombKeyFILE THEN FClose CombKeyFILE


'---- close com port if opened

CPrint "CLOSE"

END SUB

SUB HistMessage STATIC

'WindMgr 9, 66, 10, 79, 4, RevAttr, RevAttr, ""

QPrintRC "  Press ENTER   ", 9, 65, RevAttr
QPrintRC " for HIGHLIGHTS ", 10, 65, RevAttr

END SUB

SUB InsertWord (Expr AS ExpressionType) STATIC

'-- insert the last word deleted from a LIFO stack
'-- and then do a ShowExpr
```

```
'— get the number of words in the deleted word stack (LIFO$)

x = LEN(LIFO$)

IF x THEN

        Code = CVI(RIGHT$(LIFO$, 2))
        Expr.SubExpr(1).Num = Expr.SubExpr(1).Num + 1
        NumExprWords = NumExprWords + 1
        EmsGet1El ExprCodes(NumExprWords).Poly, LEN(ExprCodes(1).Poly), Code, PolySegEMS
        ExprCodes(NumExprWords).Code = Code
        SortT ExprCodes(1), NumExprWords, DESCEND, LEN(ExprCodes(1)), 2, -3
        Expr.Num = 1
        Expr.SubExpr(1).Num = NumExprWords
        FOR k = 1 TO NumExprWords
                Code2Str Expr.SubExpr(1).Phrase, k, ExprCodes(k).Code
        NEXT

        IF x = 2 THEN
                LIFO$ = ""
        ELSE
                LIFO$ = LEFT$(LIFO$, x - 2)
        END IF

ELSE

        '— no words to undelete (insert)
        Chime 1

END IF

END SUB

FUNCTION KeyInstr% (KeyStr$, Srch$) STATIC

' returns TRUE if Srch$ is in KeyStr$ at an odd offset
' otherwise FALSE (i.e. it wasn't found, or was found at an even offset)

k = INSTR(KeyStr$, Srch$)

' continue searching if it was found at an even offset
' but stop if it's 0 (not found anymore) or odd (found correctly)

DO WHILE k / 2 = k \ 2 AND k <> 0' even
        k = INSTR(k + 1, KeyStr$, Srch$)
LOOP
IF k = 0 THEN ' it wasn't found in the right place
        KeyInstr% = FALSE
ELSE
        'NOTE: Since this routine does not return a TRUE value, but simply a
        '  non-zero value, the NOT operator can't be used with this function
        '  When using KeyInstr% in an IF..THEN statement, it must test for
        '  KeyInstr% = FALSE.
        KeyInstr% = (k + 1) \ 2' return actual location
END IF

END FUNCTION

FUNCTION KeyMid$ (KeyStr$, Start) STATIC

KeyMid$ = MID$(KeyStr$, Start * 2 - 1, 2)

END FUNCTION

SUB LoadData STATIC

'————————————————————————————— D I C T I O N A R Y ———

' Load in Word Sorted Dictionary directly into EMS (translate Code# to Word$)

EmsPg = .EmsGetPFSeg%
File$ = LstDir$ + "DICT.COD"

GOSUB UpdateStatusLine

DIM DictCodeTemp AS DictType
IF NOT Exist%(File$) THEN CLS : PRINT File$; " not found.": END

SizeofFile& = FileSize&(File$)
DictCodeNum = SizeofFile& \ LEN(DictCodeTemp)

' "Loading " + Num$(DictCodeNum) + " Dictionary Synonym Entries:  .", r, c

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, DictCodeH%, File$

IF EmsError% GOTO EMSErrHandler

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)

FOpenAll File$, 0, 4, DictCodeFILE
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC STR$(i), R, c - 3, -1

        '— map pages of the DictCodeH% memory to the EMS upper mem page frame
        FOR j = 1 TO 2
```

```
            EmsMapMem DictCodeHX, j, (i - 1) * 2 + j
            IF EmsError% GOTO EMSErrHandler
        NEXT

        '-- seek to beginning of current block
        FSeek DictCodeFILE, (i - 1) * ThirtyTwoK&
        IF DosError% GOTO DOSErrHandler

        IF i < Num32kBlocks + 1 THEN
            '-- get the 32k block and put it directly into the EMS page frame
            FGetA DictCodeFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&
            IF DosError% GOTO DOSErrHandler
        ELSE
            '-- Load the left over (<32k) bytes
            FGetA DictCodeFILE, BYVAL EmsPg, BYVAL 0, LeftOver&
            IF DosError% GOTO DOSErrHandler
        END IF

NEXT

FClose DictCodeFILE

'---- contains pointers to DICT.COD in EMS for use in FullText
REDIM Array1(1 TO DictCodeNum) AS Array1Type
DictLEN = LEN(DictCodeTemp)

GOSUB UpdateStatusLine

FOR i = 1 TO DictCodeNum
        Array1(i).RecNum = i
        EmsGet1EL DictCodeTemp, DictLEN, i, DictCodeHX
        Array1(i).CodeNum = DictCodeTemp.Code
NEXT
SortT Array1(1), DictCodeNum, ASCEND, LEN(Array1(1)), 2, -1
Array2EMS Array1(1), LEN(Array1(1)), DictCodeNum, Array1EMS
ERASE Array1

' Load in Code Sorted Dictionary directly into EMS (translate Code to Word$ )

File$ = LstDir$ + "DICT.WRD"
DIM DictWordTemp AS DictType
SizeofFile& = FileSize&(File$)
DictWordNum = SizeofFile& \ LEN(DictWordTemp)
SecCode = DictWordNum - 3030
ArtCode = DictWordNum - 30

GOSUB UpdateStatusLine

NumPages = SizeofFile& \ SixteenK + 2' round off to nearest 2 pages
EmsAlloc NumPages, DictWordHX, File$
IF EmsError% GOTO EMSErrHandler

Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)

FOpenAll File$, 0, 4, DictWordFILE
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC STR$(i), R, c - 3, -1

        '-- map pages of the DictWordHX memory to the EMS upper mem page frame
        FOR j = 1 TO 2
            EmsMapMem DictWordHX, j, (i - 1) * 2 + j
            IF EmsError% GOTO EMSErrHandler
        NEXT

        '-- seek to beginning of current block
        FSeek DictWordFILE, (i - 1) * ThirtyTwoK&
        IF DosError% GOTO DOSErrHandler

        IF i < Num32kBlocks + 1 THEN
            '-- get the 32k block and put it directly into the EMS page frame
            FGetA DictWordFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&
        ELSE
            '-- Load the left over (<32k) bytes
            FGetA DictWordFILE, BYVAL EmsPg, BYVAL 0, LeftOver&
        END IF
        IF DosError% GOTO DOSErrHandler

NEXT

FClose DictWordFILE
'------------------------------ I N V E R T E D   R E L A T I V E S ----

' Load/open the Relative Inverted Files for the SWAPS routine

File$ = NdxDir$ + "REL-INVs.NDX"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF
NdxLEN = 6
FreqCompNum = FileSize&(File$) \ NdxLEN
REDIM FCIndx(1 TO DictWordNum) AS SmallNdxType

GOSUB UpdateStatusLine

FGetAH File$, FCIndx(1), NdxLEN, FreqCompNum
```

```
File$ = NdxDir$ + "RELATIVE.63"
FOpenAll File$, 0, 4, Freq63FILE


'------------------------------------------------ I N V E R T E D ----

' Load the Inverted Data Files

File$ = NdxDir$ + "KYINVRTs.NDX"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

NumKeys = FileSize&(File$) \ NdxLEN

REDIM KYIndx(1 TO DictWordNum) AS SmallNdxType

GOSUB UpdateStatusLine

FGetAH File$, SEG KYIndx(1), NdxLEN, NumKeys

File$ = NdxDir$ + "KYINVERT.DAT"
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

GOSUB UpdateStatusLine

FOpenAll File$, 0, 4, KYInvertDatFILE

'---- Open the NEWKEY.NDX file,sorted by weight*poly^0.25 for use when
'                                  showing keywords in current document
File$ = NdxDir$ + "NEWKEY.NDX"
NumRecords& = FileSize&(File$) \ 256
IF NOT ExistX(File$) THEN
        CLS
        PRINT File$; " was not found."
        CALL Freehandles
        END
END IF

GOSUB UpdateStatusLine

FOpenAll File$, 0, 4, KeyNdxFile

'------------------------------------------------ B I T C O U N T ----

DIM BitCountX(0 TO 255)
FOR i = 0 TO 255
        x$ = CHR$(i)
        x = 0
        FOR j = 0 TO 7
                x = x - GetBitX(x$, j)
        NEXT
        BitCountX(i) = x
NEXT

'------------------------------------ S I N G K E Y / C O M B K E Y ----

File$ = LstDir$ + "SINGKEY.STR"
NumSingKey = FileSize&(File$) \ 32

GOSUB UpdateStatusLine

FOpenAll File$, 0, 4, SingKeyFILE
IF DosErrorX GOTO DOSErrHandler

File$ = LstDir$ + "COMBKEY.STR"
NumCombKey = FileSize&(File$) \ 64
FOpenAll File$, 0, 4, CombKeyFILE

GOSUB UpdateStatusLine

IF DosErrorX GOTO DOSErrHandler

'------------------------------------------ P O L Y S E M Y ----------

NumPages = CLNG(FreqCompNum) * PolyLEN \ SixteenK + 2
EmsAlloc NumPages, PolySemyEMS, "Polysemy Storage"
File$ = NdxDir$ + "POLYSEMY.dat"
GOSUB UpdateStatusLine

FOpenAll File$, 0, 4, PolysemyFILE

'---- to fit in smaller memory load only as many poly values as there are
'FOR i = 1 TO FreqCompNum
'    FGetRT PolysemyFILE, PolyValue, CLNG(i), PolyLEN
'    EmsSet PolyValue, PolyLEN, CLNG(i), PolySemyEMS
'NEXT

SizeofFile& = FileSize&(File$)
Num32kBlocks = SizeofFile& \ ThirtyTwoK&
LeftOver& = SizeofFile& - (Num32kBlocks * ThirtyTwoK&)
```

```
FOR i = 1 TO Num32kBlocks + 1

        QPrintRC STR$(i), R, c - 3, -1

        '-- map pages of the PolySemyEms memory to the EMS upper mem page frame
        FOR j = 1 TO 2
                EmsMapMem PolySemyEMS, j, (i - 1) * 2 + j
                IF EmsError% GOTO EMSErrHandler
        NEXT

        '-- seek to beginning of current block
        FSeek PolysemyFILE, (i - 1) * ThirtyTwoK&
        IF DosError% GOTO DOSErrHandler

        IF i < Num32kBlocks + 1 THEN
                '-- get the 32k block and put it directly into the EMS page frame
                FGetA PolysemyFILE, BYVAL EmsPg, BYVAL 0, ThirtyTwoK&
        ELSE
                '-- load the left over (<32k) bytes
                FGetA PolysemyFILE, BYVAL EmsPg, BYVAL 0, LeftOver&
        END IF
        IF DosError% GOTO DOSErrHandler

NEXT
FClose PolysemyFILE

File$ = DocDir$ + ".NDX"
DocumNum& = (FileSize&(File$) \ 8)
Limit = SQR(DocumNum& / 300)
IF Limit < 1 THEN Limit = 1

'-------------------------------------------------------------------------

LoadedFlag = TRUE

EXIT SUB

'-------------------------------------------------------------------------

DOSErrHandler:

        Chime 4
        CLS
        PRINT "Dos Error:"; WhichError%
        Freehandles
        STOP

'-------------------------------------------------------------------------

EMSErrHandler:

        Chime 4
        CLS
        PRINT "Ems error:"; EmsError%
        Freehandles
        STOP

'-------------------------------------------------------------------------
UpdateStatusLine:

        NumFilesLoaded = NumFilesLoaded + 1
        IF Lang$ = "GERMAN" THEN
          QPrintRC " [ EINLADEN DER DATEN" + STR$(NumFilesLoaded) + " ] ", 24, 29, NormAttr
        ELSE
          QPrintRC " [ LOADING DATA FILES" + STR$(NumFilesLoaded) + " ] ", 24, 29, NormAttr
        END IF

        RETURN

END SUB

FUNCTION Num$ (x) STATIC

Num$ = LTRIM$(RTRIM$(STR$(x)))

END FUNCTION

SUB PrintAbstr (First&, Last&, FileNum, RecNum&) STATIC

'-------------------------------------------------------------------------
'show author name
'FOpenAll DocDir$ + ".NAM", 0, 4, NameFile
'IF NameFile <> -1 THEN
'   DIM Author AS str40
'   FGetRT NameFile, Author, CLNG(RecNum&), 40
'   QPrintRC "AUTHOR: " + RTRIM$(Author.Str), 1, 23, NormAttr
'END IF
'FClose NameFile
'-------------------------------------------------------------------------
LinCount = Last& - First& + 1

IF LinCount < 1 THEN LinCount = 1

REDIM AbstrText(1 TO LinCount) AS AbstrType
REDIM AbstrClean(1 TO LinCount) AS AbstrType

LenAbst = LEN(AbstrText(1))
```

```
        j = 1
        FOR i& = First& TO Last&
                GET FileNum, i&, AbstrText(j)
                Start = 1
                DO
                  a = INSTR(Start, AbstrText(j).Str, CHR$(196))
                  IF a THEN MID$(AbstrText(j).Str, a, 1) = "-": Start = a + 1
                LOOP UNTIL a = 0 OR Start > LEN(AbstrText(j).Str)
                AbstrClean(j).Str = AbstrText(j).Str
                CALL ChangeChar(AbstrClean(j).Str, New$, Keep$)

        '  vvvvvvvvvvvvvvvvvvvvvvvvvvvvv
        '  blank line between sentences

                IF ASC(AbstrText(j).Word(1)) >= 100 THEN
                        AbstrText(j).Word(1) = CHR$(ASC(AbstrText(j).Word(1)) - 100)
                        j = j + 1
                        IF j > LinCount THEN
                                LinCount = j
                                REDIM PRESERVE AbstrText(1 TO LinCount) AS AbstrType
                                REDIM PRESERVE AbstrClean(1 TO LinCount) AS AbstrType
                        END IF
                        AbstrText(j).Str = STRING$(80, 32)
                END IF
                '----------------------------------------
                j = j + 1
                IF j > LinCount THEN
                        LinCount = j
                        REDIM PRESERVE AbstrText(1 TO LinCount) AS AbstrType
                        REDIM PRESERVE AbstrClean(1 TO LinCount) AS AbstrType
                END IF
        NEXT

        LinePtr = 1       ' Set Line pointer
        PrevLinePtr = 0
        PrintAgain:
        DO
                IF LinePtr <> PrevLinePtr THEN
                        PrevLinePtr = LinePtr

                        ' Update the 24 lines of text
                        IF AscInKee <> UP AND AscInKee <> Dn THEN

                                '-- Print information bar at bottom
                        IF TeraTypeMode$ = "LOCAL" THEN
                                LeftChar$ = CHR$(26)
                                RightChar$ = CHR$(27)
                        ExpChar$ = "F2"
                        DirDocChar$ = "F5"
        ELSE
                        LeftChar$ = CHR$(LeftArrowKey)
                        RightChar$ = CHR$(RightArrowKey)
                        ExpChar$ = CHR$(ShowExprKey)
                        DirDocChar$ = CHR$(DirNumKey)
        END IF
        IF Lang$ = "GERMAN" THEN
                Infoline$ = ExpChar$ + ": Sucheingabe" + LeftChar$ + ": Naechstes Dok " + RightChar$ + ": Vorhergeh Dok " + Di
        ELSE
                Infoline$ = ExpChar$ + ": Expr  " + LeftChar$ + ": Next  " + RightChar$ + ": Prev  " + DirDocChar$ + ": DocNum
        END IF

                CALL References(Infoline$)
        END IF

        ULr = 7: ULc = 1: LRr = 23: LRc = 80
        IF TeraTypeMode$ = "LOCAL" THEN LRr = 24
        CALL ClearScr0(ULr, ULc, LRr, LRc, NormAttr)' clear bottom portion
        CALL DrawBox(ULr, ULc, LRr, LRc, 2, NormAttr)' box around the bottom portion
        IF Lang$ = "GERMAN" THEN
          QPrintRC LMarg$ + "                                                                                    " + RMarg$, 7, 3, NormA
          QPrintRC " HIGHLIGHTS (ACHTUNG: TEXT KANN IRREFUEHREN DA NICHT IM SINNZUSAMMENHANG)", 7, 5, -1
        ELSE
          QPrintRC LMarg$ + "                                                                    " + RMarg$, 7, 11, NormAttr
          QPrintRC "HIGHLIGHTS (NOTE: DO NOT RELY ON THIS TEXT OUT OF CONTEXT)", 7, 12, RevAttr
        END IF
END IF
IF TeraTypeMode$ = "LOCAL" THEN
        R = 15
ELSE
        R = 14
END IF
FOR i = 0 TO R
        ThisLine = i + LinePtr
        IF ThisLine <= LinCount THEN
                ' print line with normal attrib.
                QPrintRC AbstrText(ThisLine).Str, i + 8, 2, NormAttr
                '-- maximum of three words per line to highlight
                FOR j = 1 TO 5
                        Word = ASC(AbstrText(ThisLine).Word(j))
                        IF Word <> 0 THEN
                                '-- .Num() is number of words in the keyword to highlight
                                Lenth = ASC(AbstrText(ThisLine).Lenth(j))
                                QPrintRC QPRTrim$(MID$(AbstrClean(ThisLine).Str, Word, Lenth)), i + 8, Word + 1, RevAttr
                        ELSE
                                EXIT FOR
                        END IF
                NEXT
        END IF
```

```
NEXT
' Wait for a key to be pressed
QuitFlag = FALSE
DO
        Inkee$ = INKEY$
LOOP UNTIL LEN(Inkee$)

IF LEN(Inkee$) = 1 THEN
        AscInKee = ASC(UCASE$(Inkee$))
ELSEIF LEN(Inkee$) = 2 THEN
        AscInKee = ASC(RIGHT$(Inkee$, 1)) + 200
END IF
SELECT CASE AscInKee
        CASE UpArrowKey, UP
                IF LinePtr > 1 THEN
                        LinePtr = LinePtr - 1
                END IF
        CASE DownArrowKey, Dn
                IF LinePtr < LinCount - R - 1 THEN
                        LinePtr = LinePtr + 1
                END IF
        CASE PgUpKey, PgUp
                IF LinePtr > 1 THEN
                        LinePtr = LinePtr - R
                        IF LinePtr < 1 THEN
                                LinePtr = 1
                        END IF
                END IF
        CASE PgDnKey, PgDn
                IF LinePtr <= LinCount - R - 1 THEN
                        LinePtr = LinePtr + R
                        IF LinePtr > LinCount THEN
                                LinePtr = LinCount
                        END IF
                END IF
        CASE HomeKey, HM
                IF LinePtr > 1 THEN
                        LinePtr = 1
                END IF
        CASE EndKey, EN
                IF LinePtr < LinCount - R THEN
                        LinePtr = LinCount - R
                END IF
        CASE ESC, RightArrowKey, LeftArrowKey, DirNumKey, ShowExprKey, F2, F3, F4, F5, ZF5, ZF7
                ExitFlag = AscInKee
                QuitFlag = TRUE

        CASE CR         'to show full text for english just comment if-then out
 IF Lang$ = "GERMAN" THEN
                        ExitFlag = AscInKee
                        QuitFlag = TRUE
 END IF

        CASE NewSearchKey, F10
                IF Lang$ = "GERMAN" THEN
                        i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                        IF i$ = "J" THEN
                                ExitFlag = AscInKee
                                QuitFlag = TRUE
                        ELSE
                                GOTO PrintAgain
                        END IF
                ELSE
                        i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                        IF i$ = "Y" THEN
                                ExitFlag = AscInKee
                                QuitFlag = TRUE
                        ELSE
                                GOTO PrintAgain
                        END IF
                END IF
        CASE ELSE
END SELECT

LOOP UNTIL QuitFlag

END SUB

FUNCTION Question$ (Prompt$, Choice$, Label$) STATIC

ULr = 12: LRr = ULr
PromptLen = LEN(Prompt$) + 6' 6 leaves space for the brackets [] and space
Wid = 50 - PromptLen
ULc = Wid - 2: LRc = ULc + PromptLen - 1
DULr = ULr - 2: DLRr = LRr + 2: DULc = ULc - 3: DLRc = LRc + 2

REDIM Scr%(1 TO ArraySize%(DULr, DULc, DLRr, DLRc))

MScrnSave DULr, DULc, DLRr, DLRc, Scr%(1)
WindMgr ULr, ULc, LRr, LRc, 4, NormAttr%, NormAttr%, ""
Chime 6
QPrintRC "[ " + Label$ + " ]", ULr - 1, ULc, RevAttr%
QPrintRC Prompt$, ULr, ULc + 1, -1
QPrintRC "[ ]", ULr, ULc + LEN(Prompt$) + 2, RevAttr%

Edit$ = LEFT$(Choice$, 1)' default choice

DO
```

```
        CALL Editor(Edit$, EdLen, Scan, 0, 1, NormAttr, RevAttr, ULr, ULc + LEN(Prompt$) + 3)

        IF Scan = ESC AND Label$ <> "WARNING!" THEN
                Edit$ = ""
                EXIT DO
        ELSEIF -Scan = F10 THEN  'new search requested
                        IF Lang$ = "GERMAN" THEN
                                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                                IF i$ = "J" THEN
                                        GlobalStatus = NewSearch

                                        EXIT DO
                                END IF
                        ELSE
                                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                                IF i$ = "Y" THEN
                                        GlobalStatus = NewSearch
                                        EXIT DO
                                END IF
                        END IF
        END IF

        IF INSTR(Choice$, Edit$) = 0 THEN
                CALL Chime(2)' not a valid choice
                Edit$ = LEFT$(Choice$, 1)' default choice
        END IF

        ' note: we can test for scan=-1 (function key 1) & pop up a help window

LOOP UNTIL INSTR(Choice$, Edit$) AND Scan = CR

Question$ = Edit$

MScrnRest DULr, DULc, DLRr, DLRc, ScrX(1)
ERASE ScrX

END FUNCTION

SUB ReadEnglishText (Txt$) STATIC

IF LEN(Txt$) = 0 THEN EXIT SUB

EndOfSentence$ = ".!?"

'— replace all quotes with spaces so as not to complicate the
'   lower casing of the first words of sentences

CALL ReplaceChar(Txt$, CHR$(34), " ")
CALL Sections(Txt$)
'—— Process text, first making all initial letters
'    of each sentence lower case

Start = 1
p = 1
LENTxt = LEN(Txt$)

DO
        DO

                IF Start > LENTxt GOTO DoneLowerCase

                p = InstrTbl2(Start, Txt$, EndOfSentence$)

                IF p = 0 THEN ' no end-of-sentence punctuation was found, so exit

                        GOTO DoneLowerCase

                ELSE  '— check for a NLG (honorific/title)

                        Start = p + 2

                        FOR i = 1 TO UBOUND(NLG$)

                         IF p > LENNLG(i) THEN

                                IF LCASE$(MID$(Txt$, p - LENNLG(i), LENNLG(i))) = NLG$(i) THEN

                                        p = 0
                                        EXIT FOR

                                END IF
                         END IF
                        NEXT

                END IF

         LOOP UNTIL p

         p = p + 2
         IF p > LENTxt GOTO DoneLowerCase
         CurrChar = MidChar%(Txt$, p)

         DO WHILE CurrChar < AscUpperA OR CurrChar > ASCUpperZ
                p = p + 1
                IF p > LENTxt GOTO DoneLowerCase
                CurrChar = MidChar%(Txt$, p)
         LOOP
         MID$(Txt$, p, 1) = CHR$(CurrChar + 32)
         Start = p + 1
LOOP
```

```
DoneLowerCase:
'FirstLetter = ASC(LEFT$(LTRIM$(Txt$), 1))
'IF FirstLetter > 65 AND FirstLetter < 90 THEN
'   MID$(Txt$, 1, 1) = CHR$(FirstLetter + 32)
'END IF
New$ = " "     ' replacement string for punctuation
Old$ = ":/.-()[]"
FOR j = 91 TO 96: Old$ = Old$ + CHR$(j): NEXT
LENOld = LEN(Old$)

CALL RemCtrl(Txt$, "")'       replace all ctrl chars with blanks

' replace only SOME punctuation with spaces
FOR j = 1 TO LENOld
      CALL ReplaceChar(Txt$, MID$(Old$, j, 1), New$)
NEXT

CALL StripRange(Txt$, 33, 37, TLen)    '— strip Punctutation ! to %
Txt$ = LEFT$(Txt$, TLen)
CALL StripRange(Txt$, 39, 47, TLen)    '— strip ' to @
Txt$ = LEFT$(Txt$, TLen)
CALL StripRange(Txt$, 58, 64, TLen)    '— strip ' to @
Txt$ = LEFT$(Txt$, TLen)
CALL StripRange(Txt$, 123, 255, TLen)  '— strip High chars
Txt$ = LEFT$(Txt$, TLen)

CALL Crunch(Txt$, " ", TLen)'   crunch all multiple spaces to 1
Txt$ = LEFT$(Txt$, TLen)
Txt$ = LTRIM$(RTRIM$(Txt$))'    remove spaces from left & right

END SUB

SUB ReadGermanText (Txt$) STATIC

CALL Sections(Txt$)
CALL Lower(Txt$)'          convert all chars to lower case
CALL RemCtrl(Txt$, "")'       replace all ctrl chars with blanks

' replace only SOME punctuation with spaces
New$ = " "
Old$ = ":/.-()[]"
FOR j = 1 TO LEN(Old$)
      CALL ReplaceChar(Txt$, MID$(Old$, j, 1), New$)
NEXT

CALL StripRange(Txt$, 33, 37, TLen)' strip Punctutation ! to %
Txt$ = LEFT$(Txt$, TLen)

' Note: the range is thru chr$(96) because all the letters are lower case
' and all numbers are being stripped out too.  We've skipped over 38
' because it's the & char which is allowed

CALL StripRange(Txt$, 39, 47, TLen)
Txt$ = LEFT$(Txt$, TLen)
CALL StripRange(Txt$, 58, 96, TLen)
Txt$ = LEFT$(Txt$, TLen)


CALL StripRange(Txt$, 123, 255, TLen)' strip High chars
Txt$ = LEFT$(Txt$, TLen)

CALL Crunch(Txt$, " ", TLen)'   crunch all multiple spaces to 1
Txt$ = LEFT$(Txt$, TLen)
Txt$ = LTRIM$(RTRIM$(Txt$))'    remove spaces from left & right

END SUB

SUB References (Text$) STATIC
'— prints a centered status line on the bottom row (25th line) of the screen
Text$ = STRING$((80 - LEN(Text$)) \ 2, 32) + Text$ + STRING$((80 - LEN(Text$)) \ 2, 32)
IF LEN(Text$) < 80 THEN Text$ = Text$ + " "
IF TermTypeMode$ = "LOCAL" THEN
      R = 25
ELSE
      R = 24
END IF
IF Cnf.MonTyp = 2 THEN
      QPrintRC Text$, R, 1, RevAttr
ELSE
      QPrintRC Text$, R, 1, OneColor(1, 7)            'RevAttr
END IF

END SUB

SUB ScrSR (SR$) STATIC

STATIC WindowOpen

IF SR$ = "S" THEN

      REDIM Scr%(1 TO 2000)
      MScrnSave 1, 1, 25, 80, SEG Scr%(1)
      WindowOpen = TRUE

ELSE

      IF WindowOpen THEN
            MScrnRest 1, 1, 25, 80, SEG Scr%(1)
            ERASE Scr%
            WindowOpen = FALSE
```

```
            END IF

        END IF

        END SUB

        SUB Sections (Txt$)

        '---Look for "sections" or "articles"
        IF Lang$ = "GERMAN" THEN
                SearchStr$ = "Par"
        ELSE
                SearchStr$ = "Sec"
        END IF
        LETTER$ = "zs"
        FOR LookStep = 1 TO 2
         Start = 1
         DO
                m = INSTR(Start, Txt$, SearchStr$)
                IF m THEN
                        c$ = "0123456789XVI "
                        j = InstrTbl(m, Txt$, c$)
                        IF j THEN  'if this is not a last word
                                Word$ = MID$(Txt$, m, j - m)
                        ELSE
                                EXIT DO
                        END IF
                NumFound = 3
                IF LookStep = 1 THEN
                  IF Lang$ = "GERMAN" THEN
                        CALL FindExact(VARPTR(Paragraph$(1)), NumFound, Word$)
                  ELSE
                        CALL FindExact(VARPTR(Section$(1)), NumFound, Word$)
                  END IF
                ELSE
                  IF Lang$ = "GERMAN" THEN
                        CALL FindExact(VARPTR(Artikel$(1)), NumFound, Word$)
                  ELSE
                        CALL FindExact(VARPTR(Article$(1)), NumFound, Word$)
                  END IF
                END IF
                IF Numfound <> -1 THEN
                        k = j
                        IF MidChar(Txt$, k) = 32 THEN k = k + 1
                        DO
                          Ch$ = MID$(Txt$, k, 1)
                          k = k + 1
                        LOOP UNTIL Ch$ <> " "
        m1 = 0
             DO
                Ch$ = MID$(Txt$, k + m1 - 1, 1)
                m1 = m1 + 1
              LOOP UNTIL Ch$ < "0" OR Ch$ > "9"
              IF m1 > 1 THEN 'there is a number
                        Numb$ = MID$(Txt$, k - 1, m1 - 1)
        IF VAL(Numb$) <= 3000 THEN
                                IF LookStep = 2 AND VAL(Numb$) > 30 GOTO NextStep
                                NewWord$ = LETTER$ + Numb$
                                m1 = INSTR(k, Txt$, " ")
                                IF m1 = 0 THEN
                                  Txt$ = LEFT$(Txt$, m - 1) + NewWord$
                                ELSE
                                  Txt$ = LEFT$(Txt$, m - 1) + NewWord$ + RIGHT$(Txt$, LEN(Txt$) - m1 + 1)
                                END
        END IF
                ELSE
                        IF SearchStr$ = "Art" THEN
                         m1 = 0
                         DO
                                Ch$ = MID$(Txt$, k + m1 - 1, 1)
                                m1 = m1 + 1
                         LOOP UNTIL Ch$ = "" OR Ch$ = " " OR m1 > LEN(Txt$)
                         Numb$ = MID$(Txt$, k - 1, m1 - 1)
                         NumFound = 30
                        CALL FindExact(VARPTR(Numbers$(1)), NumFound, Numb$)
                         IF NumFound <> -1 THEN
                                NewWord$ = "za" + LTRIM$(STR$(NumFound + 1))
                                m1 = INSTR(k, Txt$, " ")
                                IF m1 = 0 THEN
                                Txt$ = LEFT$(Txt$, m - 1) + NewWord$
                                ELSE
                                Txt$ = LEFT$(Txt$, m - 1) + NewWord$ + RIGHT$(Txt$, LEN(Txt$) - m1 + 1)
                                END IF
                               END IF
                        END IF
                         END IF
                 END IF
                END IF
        NextStep:
                        Start = m + 1
                        LOOP UNTIL m = 0
         SearchStr$ = "Art"
         LETTER$ = "za"
         NEXT
        DO
          m = INSTR(Start, Txt$, CHR$(21))
          IF m THEN
                        k = m + 1
                       DO
                         Ch$ = MID$(Txt$, k, 1)
```

```
                k = k + 1
              LOOP UNTIL Ch$ <> " "
              a1 = 0
              DO
                Ch$ = MID$(Txt$, k + a1 - 1, 1)
                a1 = a1 + 1
              LOOP UNTIL Ch$ < "0" OR Ch$ > "9"
              Numb$ = MID$(Txt$, k - 1, a1 - 1)
                    IF VAL(Numb$) < 3000 THEN
                NewWord$ = "za" + Numb$
              a1 = INSTR(k, Txt$, " ")
              IF a1 = 0 THEN
                    Txt$ = LEFT$(Txt$, a - 1) + NewWord$
              ELSE
                    Txt$ = LEFT$(Txt$, a - 1) + NewWord$ + RIGHT$(Txt$, LEN(Txt$) - a1 + 1)
              END IF
                    END IF
  END IF
LOOP UNTIL a = 0
END SUB

SUB SelectMenu (Ch$, Expression AS ExpressionType, HotStr$, GlobalStatus) STATIC

SELECT CASE Ch$
      CASE MID$(HotStr$, 1, 1)'Other Suggested Words: SWAPS/Relatives
            OtherWords Expression, WordPtr
      CASE MID$(HotStr$, 3, 1)' Delete a Word
            DeleteWord Expression
      CASE MID$(HotStr$, 4, 1)' Insert Last Deleted Word
            InsertWord Expression
      CASE MID$(HotStr$, 2, 1)' Add New Words
            AddSearchTerms Expression
      CASE MID$(HotStr$, 6, 1)' New Search
            IF Lang$ = "GERMAN" THEN
                  i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                  IF i$ = "J" THEN
                        GlobalStatus = NewSearch
                  ELSE
                        GlobalStatus = 3
                  END IF
            ELSE
                  i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                  IF i$ = "Y" THEN
                        GlobalStatus = NewSearch
                  ELSE
                        GlobalStatus = 3
                  END IF
            END IF
      CASE MID$(HotStr$, 5, 1)' View records
            GlobalStatus = EditSearch
            REDIM MatchRecords(1 TO 1) AS RecInfoType
            RankRecords MatchRecords(), NumFound, Expression
            IF NumFound > 0 THEN
                  ShowDoc MatchRecords(), NumFound, Expression, RecNum
            END IF
      CASE MID$(HotStr$, 7, 1)' Exit program
            ScrSR "3"
            GlobalStatus = EX
      CASE ELSE
END SELECT

END SUB

SUB ShowAbstr (RecNum&) STATIC

DIM FileNdx AS ISAMtype

FilNum = FREEFILE
OPEN AbstrDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 8
      GET FilNum, RecNum&, FileNdx
CLOSE FilNum

FilNum = FREEFILE
OPEN AbstrDir$ + ".TXT" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 88
      CALL PrintAbstr(FileNdx.First, FileNdx.Last, FilNum, RecNum&)
CLOSE FilNum

END SUB

SUB ShowDoc (Show() AS RecInfoType, NumShow, Expr AS ExpressionType, RecNum) STATIC

REDIM ShowHeaders(1 TO NumShow) AS HeaderType
DIM KeyNdxTemp AS KeyNdxType127

PrevGlobalStatus = GlobalStatus
Curr = 1
LastFlag = FALSE
GOSUB File
DO
      SELECT CASE ExitFlag

            CASE F2, ShowExprKey ' Show the current expression
                  REDIM Scr%(1 TO 2000)
                  CALL MScrnSave(1, 1, 25, 80, SEG Scr%(1))
                  CALL ShowExpr(Expr)
                  CALL ShowQuery
                  CALL MScrnRest(1, 1, 25, 80, SEG Scr%(1))
                  ERASE Scr%
                  GOSUB File
```

```
        CASE F4, 275, LeftArrowKey                          'Previous Document
                IF Curr > 1 THEN Curr = Curr - 1
                GOSUB File

        CASE F3, 277, RightArrowKey                         'Next Document
                IF Curr < NumShow THEN Curr = Curr + 1
                GOSUB File
        CASE F5, DirNumKey                                  'Direct type a number
                 Chime 5
                 SaveCurr = Curr
                 DO
                        CLS
                        Infoline$ = "Esc:Previous Screen  Enter:To Go Directly to Entered Document Number"
                        CALL References(Infoline$)
                        IF Cnf.MonTyp = 2 THEN
                             Col = RevAttr
                        ELSE
                             Col = OneColor(14, 4)
                        END IF
                        ClearScr 2, 20, 4, 65, Col
                        CALL DrawBox(2, 20, 4, 65, 2, Col)
                        QPrintRC " ENTER DOCUMENT NUMBER [1 -" + STR$(UBOUND(Show)) + "] :", 3, 26, OneColor(14, 4)
TooBig:
                        Cur$ = "     "
                        CurLen = 4
                        CALL Editor(Cur$, CurLen, Scan, 1, 0, RevAttr, RevAttr, 3, 60)
                        IF Scan = 13 THEN
                                Curr = VAL(Cur$)
                                IF Curr > UBOUND(Show) OR Curr = 0 THEN
                                        Chime 6
                                        ClearScr 5, 25, 8, 61, Col
                                        CALL DrawBox(5, 25, 8, 61, 2, Col)
                                        QPrintRC "Number is too big or not valid.", 6, 29, Col
                                        QPrintRC "Press space bar to continue.", 7, 30, Col
                                        WaitSpace
                                        QPrintRC "    ", 3, 60, -1
                                        COLOR Fg, BG
                                ELSE
                                        COLOR Fg, BG
                                        GOSUB File
                                END IF
                        ELSE
                                IF Scan = 27 THEN
                                        Curr = SaveCurr
                                GOSUB File
                                ELSE
                                     QPrintRC "    ", 3, 60, -1
                                     GOTO TooBig
                                END IF
                        END IF
                 LOOP UNTIL (Curr <= UBOUND(Show) AND Curr > 0)

        CASE F10, NewSearchKey                              'NewSearch
                GlobalStatus = NewSearch
                HistFlag = FALSE
                EXIT SUB

        CASE CR:
                DIM FileNdx AS ISAMtype
                PrevGlobalStatus = GlobalStatus
                REDIM Scr2%(1 TO 2000)
                CALL MScrnSave(1, 1, 6, 80, SEG Scr2%(1))

                CLS
                RecNum& = Show(Curr).Rec
                FilNum = FREEFILE
                OPEN DocDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 8
                     GET FilNum, RecNum&, FileNdx
                CLOSE FilNum

                FilNum = FREEFILE
                OPEN DocDir$ + ".TXT" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 80
                     CALL FullText(FileNdx.First, FileNdx.Last, FilNum, SrchKeys$)
                CLOSE FilNum

                SELECT CASE ExitFlag

                        CASE ESC:
                                CLS
                                CALL MScrnRest(1, 1, 6, 80, SEG Scr2%(1))
                                CALL ShowAbstr(RecNum&)
                                IF ExitFlag = ESC THEN HistFlag = FALSE: GOSUB File

                        CASE 277, F3, RightArrowKey
                                IF Curr < NumShow THEN Curr = Curr + 1
                                GOSUB File
                        CASE 275, F4, LeftArrowKey
                                IF Curr > 1 THEN Curr = Curr - 1
                                GOSUB File

                END SELECT
                ERASE Scr2%

        END SELECT

LOOP UNTIL ExitFlag = ESC

EXIT SUB

'-------------------------------------------------------------------
```

```
File:
        CurrRec& = Show(Curr).Rec
        CurrRecDebug = Curr   '***rs debug
        CALL ShowKeywords(CurrRec&)
        QPrintRC "[" + LTRIM$(STR$(Curr)) + "]", 1, 75, RevAttr
'     FilNum = FREEFILE
'     OPEN DocDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 8
'     GET FilNum, CurrRec&, FileNdx
'     CLOSE FilNum
'     NumPg = (FileNdx.Last - FileNdx.First) / 60
'     IF NumPg = 1 THEN
'        QPrintRC "Doc" + STR$(CurrRec&) + "  " + " 1 page", 1, 3, NormAttr
'     ELSE
'        QPrintRC "Doc" + STR$(CurrRec&) + "  " + STR$(NumPg) + " pages", 1, 3, NormAttr
'     END IF
QPrintRC "[Doc" + STR$(CurrRec&) + "]", 1, 3, RevAttr
        IF Curr = 1 THEN CALL ShowHist(NumShow, Curr, Expr)
        IF Curr <= NumShow THEN
                IF (HistFlag <> CR) AND Curr <> 1 THEN
                        CALL ShowHist(NumShow, Curr, Expr)
                ELSE
                        IF Curr > 2 THEN
                                IF (MatchRecVals(Curr).Value < .6 * MatchRecVals(Curr - 1).Value OR MatchRecVals(Curr).Value <
                        END IF
                END IF
        END IF
'CurrRec& = CurrRec& MOD 600
        DO WHILE (HistFlag <> ESC AND HistFlag <> CR) AND (Curr >= 1)
                SELECT CASE HistFlag
                CASE 275, F4, LeftArrowKey '-- previous
                        IF Curr > 1 THEN Curr = Curr - 1
                        CurrRec& = Show(Curr).Rec
                        CurrRecDebug = Curr   '***rs debug
                        CALL ShowKeywords(CurrRec&)
                        QPrintRC "[" + LTRIM$(STR$(Curr)) + "]", 1, 75, RevAttr

'         FilNum = FREEFILE
'         OPEN DocDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 8
'         GET FilNum, CurrRec&, FileNdx
'         CLOSE FilNum
'         NumPg = (FileNdx.Last - FileNdx.First) / 60
'         IF NumPg = 1 THEN
'            QPrintRC "Doc" + STR$(CurrRec&) + "  " + " 1 page", 1, 3, NormAttr
'         ELSE
'            QPrintRC "Doc" + STR$(CurrRec&) + "  " + STR$(NumPg) + " pages", 1, 3, NormAttr
'         END IF

QPrintRC "[Doc" + STR$(CurrRec&) + "]", 1, 3, RevAttr
                        IF Curr > 24 OR FullFlag THEN
                                IF FullFlag THEN
                                        CALL ShowHist(NumShow, Curr, Expr)
                                ELSE
                                        CALL ScrollHist(Curr, "R", NumShow)
                                END IF
                        ELSE
                                CALL ReWriteHist(Curr)
                        END IF

                CASE 277, F3, RightArrowKey '-- next
                        IF Curr < NumShow THEN Curr = Curr + 1
                        CurrRec& = Show(Curr).Rec
                        CurrRecDebug = Curr   '***rs debug
                        IF NOT LastFlag THEN CALL ShowKeywords(CurrRec&)
                        QPrintRC "[" + LTRIM$(STR$(Curr)) + "]", 1, 75, RevAttr
QPrintRC "[Doc" + STR$(CurrRec&) + "]", 1, 3, RevAttr
'         FilNum = FREEFILE
'         OPEN DocDir$ + ".NDX" FOR RANDOM ACCESS READ SHARED AS FilNum LEN = 8
'         GET FilNum, CurrRec&, FileNdx
'         CLOSE FilNum
'         NumPg = (FileNdx.Last - FileNdx.First) / 60
'         IF NumPg = 1 THEN
'            QPrintRC "Doc" + STR$(CurrRec&) + "  " + " 1 page", 1, 3, NormAttr
'         ELSE
'            QPrintRC "Doc" + STR$(CurrRec&) + "  " + STR$(NumPg) + " pages", 1, 3, NormAttr
'         END IF

                        IF Curr > 25 OR FullFlag THEN
                                IF FullFlag THEN
                                        CALL ShowHist(NumShow, Curr, Expr)
                                ELSE
                                        CALL ScrollHist(Curr, "L", NumShow)
                                END IF
                        ELSE
                                CALL ReWriteHist(Curr)
                        END IF
                CASE ##, HomeKey
                        IF Curr <> 1 THEN Curr = 1
                                CurrRec& = Show(Curr).Rec
                                CurrRecDebug = Curr   '***rs debug
                                CALL ShowKeywords(CurrRec&)
                                QPrintRC "[" + LTRIM$(STR$(Curr)) + "]", 1, 75, RevAttr
                                QPrintRC "[Doc" + STR$(CurrRec&) + "]", 1, 3, RevAttr
                                CALL ShowHist(NumShow, Curr, Expr)
                CASE EN, EndKey
                        IF Curr <> NumShow THEN Curr = NumShow
                                CurrRec& = Show(Curr).Rec
                                CurrRecDebug = Curr   '***rs debug
                                CALL ShowKeywords(CurrRec&)
                                QPrintRC "[" + LTRIM$(STR$(Curr)) + "]", 1, 75, RevAttr
                                QPrintRC "[Doc" + STR$(CurrRec&) + "]", 1, 3, RevAttr
```

```
                                        CALL ShowHist(NumShow, Curr, Expr)
                                        LastFlag = TRUE
                        CASE ELSE
                                EXIT DO
                        END SELECT

                LOOP
                IF HistFlag = ESC THEN EXIT SUB
                ExitFlag = HistFlag
                IF GlobalStatus <> NewSearch AND ExitFlag = CR THEN CALL ShowAbstr(CurrRec%)
                IF ExitFlag = ESC THEN HistFlag = FALSE: GOTO File:
                RETURN
END SUB

SUB ShowExpr (Expr AS ExpressionType) STATIC

CALL ClearBG

TotSubExpr = 0
IF Expr.Num THEN

        FOR i = 1 TO Expr.Num
                TotSubExpr = TotSubExpr + Expr.SubExpr(i).Num
        NEXT

ELSE ' no search expression has been entered, so display a note to the user

        Note$ = "No Search Expression Yet"
        ULr = 2: ULc = 2: Frame = 1: LRr = ULr + 3: LRc = ULc + LEN(Note$) + 6
        CALL WindMgr(ULr + 1, ULc + 1, LRr + 1, LRc + 1, Frame, NormAttr, RevAttr, "")
        CALL QPrintRC(Note$, ULr + 2, ULc + 2, -1)
        EXIT SUB

END IF

IF TotSubExpr > 17 THEN ' can't display more than 17 on the screen
        IF Lang$ = "GERMAN" THEN
          DispMsg "Zuviele Wortel System verarbeitet maximal 20 Suchbegriffe", 0, 0
        ELSE
          DispMsg "Cannot process: You have too many words in your search expression.", 0, 0
        END IF
        CALL WaitSpace
        CALL Chime(3)
        EXIT SUB
END IF
Old = Expr.SubExpr(1).Num
REDIM p$(1 TO Expr.Num, 1 TO Old)
MaxWordLen = 34: LRr = 2
'-- find maximum word length
FOR S = 1 TO Expr.Num
        IF Expr.SubExpr(S).Num > Old THEN REDIM PRESERVE p$(1 TO Expr.Num, 1 TO Expr.SubExpr(S).Num)
        FOR i = 1 TO Expr.SubExpr(S).Num

          LRr = LRr + 1
          c = Str2Code%(Expr.SubExpr(S).Phrase, i)
          p$(S, i) = Dict$(c)
          IF LEN(p$(S, i)) > MaxWordLen THEN MaxWordLen = LEN(p$(S, i))
        NEXT
NEXT

' draw the Expression box only if FindExpr was called
' (i.e. searched on the whole expression)
IF Expr.Match >= 0 THEN ' it wasn't -1 so FindExpr was called

        ULr = 2: ULc = 2: Frame = 219
        LRc = ULc + MaxWordLen + 2

END IF

ULr = 3
ULc = 3
Frame = 1
LRc = ULc + MaxWordLen ' -1
n$ = "/Docs"
CALL WindMgr(ULr + 1, ULc + 1, LRr + 1, LRc + 1, Frame, NormAttr%, NormAttr%, n$)
FOR S = 1 TO Expr.Num
        FOR i = 1 TO Expr.SubExpr(S).Num
                IF p$(S, i) > "" THEN QPrintRC p$(S, i), ULr + i, ULc + 1, -1
        NEXT
NEXT

END SUB

SUB ShowKeywords (RecNum&) STATIC
'---- 4 rows of 80 columns, including window
ULr = 1: ULc = 1: LRr = 6: LRc = 80: LenCol = LRr - ULr - 1: n = 0
'CLS
CALL ClearScrO(ULr, ULc, LRr, LRc, RevAttr)' clear bottom portion
CALL DrawBox(ULr, ULc, LRr, LRc, 2, RevAttr)

'---- now display keywords

DIM KeyNdxTemp AS KeyNdxType127

'-- get 63-word list from "newkey.ndx"
FGetRT KeyNdxFile, KeyNdxTemp, RecNum&, LEN(KeyNdxTemp)
IF Lang$ = "GERMAN" THEN
  CALL QPrintRC(LMarg$ + " WICHTIGSTE BEGRIFFE: " + RMarg$, ULr, 28, -1)
ELSE
```

```
    CALL QPrintRC(LMarg$ + " KEY WORDS: " + RMarg$, ULr, 32, -1)
  END IF
  KeepExpr$ = ""
  '------ debug
  'QPrintRC STR$(RecNum&) + STR$(KeyNdxTemp.Num), 1, 1, -1
  'IF KeyNdxTemp.Num > 32767 OR KeyNdxTemp.Num < 1 THEN LOCATE 18, 1: BEEP: STOP
  '------ debug
  REDIM PRESERVE KeyWords(1 TO KeyNdxTemp.Num) AS WordShowType
  FOR i = 1 TO KeyNdxTemp.Num
        'mark with '*' words that match the query
        ks = KeyInstrX(ExprKeys$, KeyMid$(KeyNdxTemp.Str, i))
        IF ks > 0 THEN
              KeyWords(i).Flag = "*"
        ELSE
              KeyWords(i).Flag = " "
        END IF
              'get word and cut it up to 22 letters + '..'
              x$ = Dict$(Str2CodeX(KeyNdxTemp.Str, i))
              x$ = RIGHT$(x$, LEN(x$) - 5)
              CALL CutWord(x$)
              KeyWords(i).Word = x$
              KeyWords(i).Code = Str2CodeX(KeyNdxTemp.Str, i)
  NEXT
  IF KeyNdxTemp.Num > 12 THEN
        'sort rest of kw by flag,i.e. push matched words up
  SortT KeyWords(13), KeyNdxTemp.Num - 12, 1, LEN(KeyWords(1)), 0, 1
  'look for how many word we should replace to first-12 part
  SearchT2 KeyWords(13), KeyNdxTemp.Num - 12, " ", Found, 0, 0, LEN(KeyWords(1)), 0, 1
  'if "found" last words are matched shift them deLeting last non-matched words before them
  DO
        n = 0
        'look should we shift any word or we have enough room on the bottom
        SearchT2 KeyWords(12), Found, "*", Shift, 1, 0, LEN(KeyWords(1)), 0, 1
        IF Shift > -1 THEN  'we should shift
              FOR i = 12 - Found TO 1 STEP -1 'look for non-matched word for replacement
                    IF KeyWords(i).Flag = " " THEN
                          FOR j = 1 TO n. 'shift all words before that which should be shifted
                                KeyWords(i) = KeyWords(i + 1)
                                i = i + 1
                          NEXT
                          KeyWords(12 - Found) = KeyWords(12 - Shift) 'shift kw
                          KeyWords(12 - Shift).Flag = " "
                          EXIT FOR
                    ELSE
                          n = n + 1.  'how many words should be shifted
                    END IF
                  NEXT
              END IF
        LOOP UNTIL Shift = -1 OR n >= 12 - Found
        'check could we shift everything. If not, shift it and decrease Found
        DO
              SearchT2 KeyWords(12), Found, "*", Shift, 1, 0, LEN(KeyWords(1)), 0, 1
              IF Shift > -1 THEN
                    Found = Found - 1
                    'look for the first word which can be shifted
                    SearchT2 KeyWords(1), 12, " ", FirstSpace, 0, 0, LEN(KeyWords(1)), 0, 1
                    IF FirstSpace > -1 THEN
                          FirstSpace = FirstSpace + 1  'real space
                          IF FirstSpace < 12 - Shift THEN 'if there is where to shift it, do it
                                KeyWords(FirstSpace) = KeyWords(12 - Shift)
                                KeyWords(12 - Shift).Flag = " "
                          END IF
                    END IF
              END IF
        LOOP UNTIL Shift = -1
        FOR i = 1 TO Found
                KeyWords(13 - i) = KeyWords(12 + Found - i + 1)'replace matched keywords beyond 12 to 12
        NEXT
  END IF
  CurrCol = ULc - 1
n = UBOUND(KeyWords)
  IF KeyNdxTemp.Num < 12 THEN
        REDIM PRESERVE KeyWords(1 TO 12) AS WordShowType
        FOR i = KeyNdxTemp.Num + 1 TO 12
              KeyWords(i).Word = " "
        NEXT
  END IF
  FOR i = 1 TO 4

'-----------***rs debug vvvvvvvvv
'------ get Salton Value for this word  ***rs
'   Code = KeyWords(i).Code
'    if Code > 0 then
'      FOR Index& = KYIndx(Code).Index TO KYIndx(Code).Index + KYIndx(Code).Num - 1
'        FGetRT KYInvertDatFILE, KYInfo, Index&, KYInfoLEN
'        IF KYInfo.Rec = RecNum& THEN
'          SaltonValue = KYInfo.Value
'          EXIT FOR
'        END IF
'      NEXT Index&
'      QPrintRC LEFT$(Dict$(Code), 15) + " S:" + STR$(SaltonValue), ULr + i, CurrCol + 2, -1'***rs debug
'    end if
'--------***rs debug ----------
        IF KeyWords(i).Flag <> " " THEN
              QPrintRC KeyWords(i).Word, ULr + i, CurrCol + 2, NormAttr
        ELSE
              QPrintRC KeyWords(i).Word, ULr + i, CurrCol + 2, -1
        END IF
```

```
        '----***rs debug vvvvvvvvvv
'--- get Salton Value for this word  ***rs
    Code = KeyWords(i + 4).Code
    if Code > 0 then
       FOR Index& = KYIndx(Code).Index TO KYIndx(Code).Index + KYIndx(Code).Num - 1
          FGetRT KYInvertDatFILE, KYInfo, Index&, KYInfoLEN

          IF KYInfo.Rec = RecNum&% THEN
             SaltonValue = KYInfo.Value
             EXIT FOR
          END IF
       NEXT Index&
       QPrintRC LEFT$(Dict$(Code), 15) + " S:" + STR$(SaltonValue), ULr + i, CurrCol + 25 + 2, -1'***rs debug
    end if
'-------***rs debug ----------
                IF KeyWords(i + 4).Flag <> " " THEN
                        QPrintRC KeyWords(i + 4).Word, ULr + i, CurrCol + 27, NormAttr
                ELSE
                        QPrintRC KeyWords(i + 4).Word, ULr + i, CurrCol + 27, -1
                END IF
'-------***rs debug vvvvvvvvvv
'---- get Salton Value for this word  ***rs
    Code = KeyWords(i + 8).Code
    if Code > 0 then
       FOR Index& = KYIndx(Code).Index TO KYIndx(Code).Index + KYIndx(Code).Num - 1
          FGetRT KYInvertDatFILE, KYInfo, Index&, KYInfoLEN
          IF KYInfo.Rec = RecNum& THEN
             SaltonValue = KYInfo.Value
             EXIT FOR
          END IF
       NEXT Index&
       QPrintRC LEFT$(Dict$(Code), 15) + " S:" + STR$(SaltonValue), ULr + i, CurrCol + 50 + 2, -1'***rs debug
    end if
'--------***rs debug ----------
                IF KeyWords(i + 8).Flag <> " " THEN
                        QPrintRC KeyWords(i + 8).Word, ULr + i, CurrCol + 52, NormAttr
                ELSE
                        QPrintRC KeyWords(i + 8).Word, ULr + i, CurrCol + 52, -1
                END IF
        NEXT
'-------***rs debug vvvvvvvvvv
   QPrintRC "Document Rank:" + STR$(RecsDebug&(CurrRecDebug)), 6, 1, -1'***rs debug
      CurrRow = CurrRow + 1
         IF CurrRow = ULr THEN
            IF CurrCol = OldCurrCol OR CurrCol = OldCurrCol + 25 THEN '-- move over to next column
               CurrRow = OldCurrRow
               CurrCol = CurrCol + 25
            ELSE '-- we're already at the bottom of the 3nd column so we're done
               EXIT FOR
            END IF
         END IF
'--------***rs debug ----------

END SUB

SUB ShowQuery

REDIM ScrX(1 TO 800)
MScrnSave 20, 1, 25, 80, ScrX(1)
ClearScr0 20, 2, 24, 79, NormAttr
CALL DrawBox(20, 2, 24, 79, 1, NormAttr)
FOR i = 1 TO 3
        IF LEN(Sentence$(i)) THEN QPrintRC Sentence$(i), 20 + i, 3, -1
NEXT
CALL References("PRESS ANY KEY TO CONTINUE")
DO
 Ch$ = INKEY$
LOOP UNTIL LEN(Ch$) > 0
MScrnRest 20, 1, 25, 80, ScrX(1)
ERASE ScrX

END SUB

FUNCTION SpaceNum$ (x, SpaceX) STATIC

SpaceNum$ = RIGHT$(STRING$(SpaceX, " ") + Num$(x), SpaceX)

END FUNCTION

FUNCTION Str2CodeX (Store$, LocationX) STATIC

Str2CodeX = CVI(MID$(Store$, LocationX * 2 - 1, 2))

END FUNCTION

SUB WaitSpace STATIC
CALL ClearBuf
DO
        Ky$ = INKEY$
        IF Ky$ = " " THEN
                EXIT SUB
        ELSEIF Ky$ = CHR$(0) + CHR$(F10 - 200) OR Ky$ = CHR$(NewSearchKey) THEN
                IF Lang$ = "GERMAN" THEN
                        i$ = Question$("NEUE SUCHE ? [J/N] ", "JN", "ACHTUNG !")
                        IF i$ = "J" THEN
                          GlobalStatus = NewSearch
                          EXIT SUB
                        END IF
                ELSE
```

```
                        i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                        IF i$ = "Y" THEN
                          GlobalStatus = NewSearch
                          EXIT SUB
                        END IF
                    END IF
                END IF
            LOOP

            END SUB

            SUB WindMgr (ULRow, ULCol, LRRow, LRCol, Frame, BoxColr, TextColr, Text$) STATIC

            CALL ClearScrO(ULRow, ULCol, LRRow, LRCol, BoxColr)
            CALL DrawBox(ULRow - 1, ULCol - 1, LRRow + 1, LRCol + 1, Frame, BoxColr)
            IF Text$ = "#Docs" THEN
                CALL QPrintRC(Text$, ULRow - 1, ULCol, TextColr)
            ELSE
                IF LEN(Text$) THEN CALL QPrintRC("[" + Text$ + "]", ULRow - 1, ULCol + 1, TextColr)
            END IF

            END SUB

            SUB WordParse (Txt$, WordList$(), NumWords) STATIC

            NumWords = 0

            TotW = InCount%(Txt$, " ") + 1' number of words in current line

            FOR Word = 1 TO TotW

                CALL Extract(Txt$, " ", Word, Start, Slen)

                IF Slen > 0 THEN

                    '---- extract word
                    w$ = MID$(Txt$, Start, Slen)

                    '---- fill out 1 and 2 char words with /'s
                    IF Slen < 3 THEN w$ = w$ + STRING$(3 - Slen, "/")

                    '---- allow only words that start with alphabetic chars "a"-"z"
                    ASCw = ASC(w$)

                    IF (ASCw >= AscA AND ASCw <= ASCZ) OR (ASCw >= AscUpperA AND ASCw <= ASCUpperZ) THEN

                        NumWords = NumWords + 1

                        '---- the following doesn't apply to GERMAN
                        IF Lang$ = "ENGLISH" THEN
                            IF RIGHT$(w$, 2) = "'s" THEN     '-- remove the 's
                                w$ = LEFT$(w$, Slen - 2)
                            ELSEIF RIGHT$(w$, 1) = "'" THEN  '-- and any final '
                                w$ = LEFT$(w$, Slen - 1)
                            END IF
                        END IF

                        '---- store the word
                        REDIM PRESERVE WordList$(1 TO NumWords)
                        WordList$(NumWords) = w$

                    END IF
                END IF

            NEXT' word in line

            END SUB
FUNCTION ZeroNum$ (x, Zero) STATIC

'-- fill a number with leading zeros

ZeroNum$ = RIGHT$(STRING$(Zero, "0") + Num$(x), Zero)

END FUNCTION

DECLARE SUB SortSwapEMS (HandleX, NumElsX)
DECLARE SUB DrawBox (ULRowX, ULColX, LRRowX, LRColX, FrameX, ColX)
DEFINT A-Z

TYPE WordType
        Word     AS STRING * 64
        LineNumb AS INTEGER
END TYPE

DEFINT A-Z

CONST FALSE = 0, TRUE = NOT FALSE, ASCEND = 0, Descend = 1
CONST MaxShow = 50

'scan code + 200 for not to mix with letters
CONST UP = 272, PGUP = 273, Dn = 280, PGDN = 281, HM = 271, EN = 279
CONST CtrlPgUp = 332, CtrlPgDn = 318, CtrlHM = 319, CtrlEN = 317
CONST F1 = 259, F2 = 260, F3 = 261, F4 = 262, F5 = 263
CONST F6 = 264, F7 = 265, F8 = 266, F9 = 267, F10 = 268

CONST ESC = 27, CR = 13

CONST NewSearch = 1, AddWords = 2, EditSearch = 3, Back = 4, Forward = 5, SWAPS = 6
```

```
'$INCLUDE: '\\vadim\c-drive\user\include\defcnf.bi'
'$INCLUDE: '\\vadim\c-drive\user\include\types.bi'
'$INCLUDE: '\\vadim\c-drive\user\INCLUDE\shared.bi'
'$INCLUDE: '\\vadim\c-drive\user\EXTERN.BAS'

DECLARE SUB AddSwaps (Expr AS ExpressionType, Exclude$, Collect() AS CollectType, NumCollect%)
DECLARE SUB BuildCombTable (Mode$)
DECLARE SUB Code2Str (Store$, Location%, Code%)
DECLARE SUB CPrint (x$)
DECLARE SUB DispMsg (Msg$, R%, c%)
DECLARE SUB EmsAlloc (NumPages%, Handle%, LoadFile$)
DECLARE SUB FindRelatives (Expr AS ExpressionType, SubNum%, Ptr%, Rel() AS CollectType, NumRel%, Exclude$)
DECLARE SUB HistMessage ()
DECLARE SUB OtherWords (Expr AS ExpressionType, WordPtr)
DECLARE SUB PickChoice (m() AS CollectType, NumM%, Label$, Pick%(), NumPick%, ExactFlag, Expr AS ExpressionType)
DECLARE SUB PickList (Items$(), Picked%(), NPicked%, Cnf AS ANY, Expression AS ANY, NewSearchKey, ShowExprKey, HK, EK, PgUK, PgDK, UK,
DECLARE SUB RankRecords (Recs() AS RecInfoType, NumFound%, E AS ExpressionType)
DECLARE SUB References (Text$)
DECLARE SUB ScrSR (SR$)
DECLARE SUB ShowExpr (Expr AS ANY)
DECLARE SUB SortEMSRankInfo (Handle%, Num&)
DECLARE SUB TermMatch (Term$, Expr AS ExpressionType, m() AS CollectType, NumM%, Repeate, ExactFlag)
DECLARE SUB VertMenu (Item$(), Choice%, MaxLen%, BoxBot%, Ky$, Action%, Cnf AS Config, Mode$, HK, EK, PgUK, PgDK, UK, DK, TermTypeMode
DECLARE SUB WaitSpace ()
DECLARE SUB WindMgr (ULRow%, ULCol%, LRRow%, LRCol%, Frame%, BoxColr%, TextColr%, Text$)
DECLARE FUNCTION BoxInput$ (Edit$, Title$, Prompt$, Row%, Col%, Scan)
DECLARE FUNCTION ComboSum& (Bit$, Value&, Mode$, Poly!())
DECLARE FUNCTION Dict$ (Code%)
DECLARE FUNCTION FirstLast% (Word$, First%, Last%, KeyType%)
DECLARE FUNCTION KeyInstr% (KeyStr$, Srch$)
DECLARE FUNCTION KeyMid$ (KeyStr$, Start)
DECLARE FUNCTION Num$ (x%)
DECLARE FUNCTION Question$ (Prompt$, Choice$, Label$)
DECLARE FUNCTION SelectRelatives% (Expr AS ExpressionType, CurrentSub%, TopKeys() AS CollectType, NumTopKeys%)
DECLARE FUNCTION SpaceNum$ (x%, SPACE%)
DECLARE FUNCTION Str2Code% (s$, k%)

'$INCLUDE: '\\vadim\c-drive\user\include\prefixes.bi'

SUB AddSearchTerms (Expr AS ExpressionType) STATIC

Repeate = FALSE
CurrentSub = 1
GlobalStatus = EditSearch

IF Expr.SubExpr(1).Num = 15 THEN ' can't add any more words
        Chime 4
        IF Lang$ = "GERMAN" THEN
                DispMsg "Fehler:  Maximal 15 Suchbegriffe.  LEERTASTE um weiterzumachen!", R, c
        ELSE
                DispMsg "ERROR: Limit of 15 Search Terms. Press the Space Bar to continue:", R, c
        END IF
        WaitSpace
        DispMsg "", 0, 0
        EXIT SUB
END IF
IF Lang$ <> "GERMAN" THEN
        CALL References("ENTER WORD OR PHRASE AND THEN PRESS ENTER OR PRESS ESC TO CANCEL")
END IF
'---- Get search term(s) from user
GetSearchTerm:

ScrSR "S"
Title$ = "Adding Search Words"
Row = 19
Term$ = BoxInput$(SPACE$(40), Title$, "Search Word/Phrase:", Row, 8, Scan)
ScrSR "R"
IF (ABS(Scan) + 200 = F10 AND Scan < 0) OR Scan = NewSearchKey THEN
                        IF Lang$ = "GERMAN" THEN
                                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                                IF i$ = "J" THEN
                                        GlobalStatus = NewSearch
                                        EXIT SUB
                                END IF
                        ELSE
                                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                                IF i$ = "Y" THEN
                                        GlobalStatus = NewSearch
                                        EXIT SUB
                                END IF
                        END IF
END IF
IF Term$ = "" THEN EXIT SUB

'---- Collect all word-code numbers matching the partial term from
'     the Dictionary (DictWord)

'---- allocate dummy space for the matching codes
REDIM MatchCodes(1 TO 1) AS CollectType

'---- collect the matching codes into the array
CALL TermMatch(Term$, Expr, MatchCodes(), NumMatchCodes, Repeate, ExactFlag)

IF NumMatchCodes = 0 THEN ' no matching entries in dictionary
        IF Repeate THEN
                IF Lang$ = "GERMAN" THEN
                        DispMsg "Das Wort '" + Term$ + "' ist bereits eingegeben. LEERTASTE um anderen Suchbegriff einzugeben!", R, c
                ELSE
                        DispMsg "Word '" + Term$ + "' is already chosen.  Press the Space Bar to enter a new Search Term.", R, c
                END IF
```

```
        ELSE
                IF Lang$ = "GERMAN" THEN
                        DispMsg "Kein Begriff gefunden der '" + Term$ + "' enthaelt. LEERTASTE um anderen Suchbegriff einzugeben!", R,
                ELSE
                        DispMsg "No words or phrases that contain '" + Term$ + "' were found.  Press the Space Bar to enter a new Sear
                END IF
        END IF
        WaitSpace
        DispMsg "", 0, 0
        ShowExpr Expr
        GOTO GetSearchTerm' get a new word to search for
END IF
IF NumMatchCodes > 1 THEN
'---- Get one or more choices from the list from the substring dict match
'  Note: the resulting list of matches is stored in word-code form
'        therefore, the PickChoice routine must convert all the
'        codes to words in a temporary array for the PickList routine

Label$ = "Select Search Term(s)"

GlobalStatus = AddWords

REDIM PickedX(15 - Expr.SubExpr(CurrentSub).Num)
CALL PickChoice(MatchCodes(), NumMatchCodes, Label$, PickedX(), NumPicked, ExactFlag, Expr)

ELSE
 NumPicked = 1
 REDIM Picked(1 TO 1)
 Picked(1) = 1 'MatchCodes(1).Code
END IF

IF NumPicked = -10 THEN ' F10 was pressed: New Search
                        IF Lang$ = "GERMAN" THEN
                                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                                IF i$ = "J" THEN
                                        GlobalStatus = NewSearch
                                        EXIT SUB
                                END IF
                        ELSE
                                        i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                                        IF i$ = "Y" THEN
                                                GlobalStatus = NewSearch
                                                EXIT SUB
                                        END IF
END IF

ELSEIF NumPicked > 0 THEN ' add words to expression

        '---- store choices into current SubExpr.Phrase

        NumPhrase = Expr.SubExpr(CurrentSub).Num
        Expr.SubExpr(CurrentSub).Num = NumPhrase + NumPicked
        FOR i = 1 TO NumPicked
                IF KYIndex(MatchCodes(Picked(i)).Code).Num THEN
                        NumExprWords = NumExprWords + 1
                        REDIM PRESERVE ExprCodes(1 TO NumExprWords) AS CodePolyType
                        ExprCodes(NumExprWords).Code = MatchCodes(Picked(i)).Code
                        EmsGet1El ExprCodes(NumExprWords).Poly, LEN(ExprCodes(1).Poly), ExprCodes(NumExprWords).Code, PolySemyEMS
                ELSE
                        IF Lang$ = "GERMAN" THEN
                                DispMsg "Der Begriff '" + Term$ + "' erscheint in keinem Dokument. LEERTASTE um anderen Begriff einzug
                        ELSE
                                DispMsg "No documents that contain '" + Term$ + "' were found.  Press the Space Bar to enter a new Sea
                        END IF
                        WaitSpace
                        DispMsg "", 0, 0
                        GOTO GetSearchTerm      ' get a new word to search for
                END IF
        NEXT
        OriginalExpr$ = ""
        SortT ExprCodes(1), NumExprWords, Descend, LEN(ExprCodes(1)), 2, -3
        FOR k = 1 TO NumExprWords
                Code2Str Expr.SubExpr(CurrentSub).Phrase, k, ExprCodes(k).Code
                Origina.Expr$ = OriginalExpr$ + MKI$(ExprCodes(k).Code)
        NEXT
        Expr.Match = -1' reset search flag in full expression
                                                ' when this flag is -1, the progras
                                                ' knows that no valid search has been done

ELSE ' none were picked

        ScrSR "R"
        GOTO GetSearchTerm

END IF


ERASE MatchCodes, Picked

Expr.Num = CurrentSub ' set number of expressions to the current sub

END SUB

SUB AddSwaps (Expr AS ExpressionType, Exclude$, Collect() AS CollectType, NumCollect) STATIC

GlobalStatus = EditSearch
```

```
IF Expr.SubExpr(1).Num = 15 THEN ' can't add any more words
        Chime 4
        DispMsg "ERROR: Limit of 15 Search Terms. No more Search Terms can be added. Press the Space Bar to continue:", R, c
        WaitSpace
        DispMsg "", 0, 0
        EXIT SUB
END IF
DIM FC63 AS FreqComp63
FC63LEN = LEN(FC63)
'— if not enough words to run SWAPS then return 12 relatives
IF Expr.SubExpr(1).Num < 2 THEN

        Code = Str2Code%(Expr.SubExpr(1).Phrase, 1)
        FGetRT Freq63FILE, FC63, CLNG(Code), FC63LEN
        i = 0: NumCollect = 0

        DO
                ExclFlag =.FALSE
                i = i + 1
                '— do not include words from Exclude$
                FOR j = 1 TO LEN(Exclude$) / 2
                        IF Str2Code(Exclude$, j) = FC63.Comp(i) THEN
                                ExclFlag = TRUE
                                EXIT FOR
                        END IF
                NEXT
                IF NOT ExclFlag THEN
                        NumCollect = NumCollect + 1
                        REDIM PRESERVE Collect(1 TO NumCollect) AS CollectType
                        Collect(NumCollect).Code = FC63.Comp(i)
                END IF
        LOOP UNTIL NumCollect = 20 OR NumCollect = FC63.Num

        EXIT SUB

END IF

'IdealFreq! = (FileSize&(DocDir$ + ".NDX") \ 8) * .007
DIM SynthTemp AS BitValue
LenSynth = LEN(SynthTemp)
FreeSpace& = FRE(-1)
'if there is enough memory then use low-memory othervays use EMS
MemReq& = CLNG(DictWordNum) * LenSynth
IF MemReq& + 1024 < FreeSpace& THEN     'leave 1K free
        REDIM Synth(1 TO DictWordNum) AS BitValue
        MemFlag = TRUE
ELSE
        NumPages = MemReq& \ SixteenK + 1
        EmsAlloc NumPages, SynthEMS, "Swaps EMS Storage"
'— clear (to 0) the EMS memory we allocated
        EMSPF = EmsGetPFSeg%
        FOR i = 1 TO NumPages
                CALL EmsMapMem(SynthEMS, 1, i)    ' map logical page i to physical page 0
                CALL InitMem(EMSPF, 0, 8192, 0)  ' clear physical page of memory to 0
        NEXT
        MemFlag = FALSE
END IF
'———————— S W A P S   R O U T I N E ————————
IF Lang$ = "GERMAN" THEN
        DispMsg "System analysiert Eingabe auf verwandte Themen", R, c
ELSE
        DispMsg "System is analyzing query for related topics", R, c
END IF

'CPrint CHR$(CR)
'CPrint " Code Freq Word/Phrase    # Relatives" + CHR$(CR)
DIM Freq63 AS FreqComp63
FCLEN = LEN(Freq63)
NumSynth = 0
FOR j = 1 TO Expr.SubExpr(1).Num
        Code = Str2Code(Expr.SubExpr(1).Phrase, j)
        FGetRT Freq63FILE, Freq63, CLNG(Code), FC63LEN
        FOR k = 1 TO Freq63.Num
                '— sum all percentage values of words in FC List (really Relative list)
                '   which contain words from the expression

                '**** boost SWAPS 1/21/92 VN
                IF KeyInstr%(OriginalExpr$, MKI$(Code)) = 0 THEN
                        AscFCInfoValue = ASC(Freq63.Value(k)) * Boost!
                ELSE
                        AscFCInfoValue = ASC(Freq63.Value(k))
                END IF
                IF MemFlag THEN
                        Synth(Freq63.Comp(k)).Value = Synth(Freq63.Comp(k)).Value + AscFCInfoValue
                        '— set bit to indicate that this word now has a value
                        SetBit Synth(Freq63.Comp(k)).Bit, j, 1
                        '— save the code number
                        Synth(Freq63.Comp(k)).Code = Freq63.Comp(k)
                ELSE
                        EmsGet1El SynthTemp, LenSynth, Freq63.Comp(k), SynthEMS
                        SynthTemp.Value = SynthTemp.Value + AscFCInfoValue
                        '— set bit to indicate that this word now has a value
                        SetBit SynthTemp.Bit, j, 1
                        '— save the code number
                        SynthTemp.Code = Freq63.Comp(k)
                        EmsSet1El SynthTemp, LenSynth, Freq63.Comp(k), SynthEMS
                END IF
        NEXT
NEXT
```

```
'---- Generate final values for words
'Debug = TRUE
QueryLen = UBOUND(ExprCodes)
REDIM Poly!(1 TO QueryLen)
FOR i = 1 TO QueryLen
        Poly!(i) = (ExprCodes(i).Poly / PolyAvg!) ^ .25
NEXT

FOR i = 1 TO DictWordNum
        'get value
        IF MemFlag THEN
                LSET SynthTemp = Synth(i)
        ELSE
                EmsGetIEl SynthTemp, LenSynth, i, SynthEMS
        END IF
        'make final calculation
        IF SynthTemp.Value THEN
                EmsGet PolyValue, PolyLEN, CLNG(SynthTemp.Code), PolySegEMS
                Temp& = SynthTemp.Value * CLNG(PolyValue.Value)
                Temp& = ComboSum&(SynthTemp.Bit, Temp&, "S", Poly!())
'vvvv
```

```
EditLen = LEN(Edit$)
UlRow = RowX: UlCol = ColX
LRRow = UlRow + 4: LRCol = UlCol + EditLen + 5
IF LEN(Prompt$) THEN LRCol = LRCol + LEN(Prompt$) + 1

CALL ClearScrO(UlRow, UlCol, LRRow, LRCol, HiAttr)
CALL DrawBox(UlRow, UlCol, LRRow, LRCol, 2, HiAttr)
IF Title$ <> "" THEN
        CALL QPrintRC(LMarg$ + Title$ + RMarg$, UlRow, (UlCol + LRCol - LEN(Title$) - 2) \ 2, HiAttr)
END IF

IF LEN(Prompt$) THEN CALL QPrintRC(Prompt$, UlRow + 2, UlCol + 2, HiAttr)
CALL DrawBox(UlRow + 1, LRCol - EditLen - 3, LRRow - 1, LRCol - 2, 1, HiAttr)

LOCATE LRRow - 2, LRCol - EditLen - 2

' Enlarge the cursor so it's easier to spot
SELECT CASE Monitor%
        CASE IS < 3'            Mono/Herc
                LOCATE , , 1, 10, 13
        CASE IS = 3'            CGA
                LOCATE , , 1, 7, 8
        CASE ELSE'              VGA/EGA/MCGA
                LOCATE , , 1, 11, 14
END SELECT

CALL Editor(Edit$, EditLen, Scan, 0, 0, HiAttr, NormAttr, LRRow - 2, LRCol - EditLen - 2)

LOCATE , , 0

BoxInput$ = RTRIM$(LEFT$(Edit$, EditLen))

END FUNCTION

SUB BuildCombTable (Mode$) STATIC

REDIM CombTable(12 TO 240) AS SINGLE

DIM DocFreqA AS LONG, DocFreqB AS LONG

IF Mode$ = "S" THEN
        Divisor! = 42!
ELSE
        Divisor! = 28!
END IF

NumKeys = LEN(ExprKeys$) \ 2
REDIM Freq63(1 TO NumKeys) AS FreqComp63
FCLEN = LEN(Freq63(1))
'-- only load the relative lists for the words in the expression

FOR i = 1 TO NumKeys
        Code = Str2Code%(ExprKeys$, i)
        FGetRT Freq63FILE, Freq63(i), CLNG(Code), FCLEN
NEXT
R = 0
FOR i = 1 TO NumKeys - 1
        FOR j = i + 1 TO NumKeys
                DocFreqA = KYIndx(Freq63(i).Code).Num
                DocFreqB = KYIndx(Freq63(j).Code).Num
                RootofDocFreq! = SQR((DocFreqA + DocFreqB) / 2)
                Subscript = i * 15 + j
                FOR k = 1 TO Freq63(j).Num
                        '-- check if A was found as a relative in B's list
                        IF Freq63(i).Code = Freq63(j).Comp(k) THEN
                                '-- the relative value for the AB pair is
                                CombTable(Subscript) = ASC(Freq63(j).Value(k))
                                EXIT FOR
                        END IF
                NEXT
                IF CombTable(Subscript) = 0 THEN
                        FOR k = 1 TO Freq63(i).Num
                                '-- check if A was found as a relative in B's list
                                IF Freq63(j).Code = Freq63(i).Comp(k) THEN
                                        CombTable(Subscript) = ASC(Freq63(i).Value(k))
                                        EXIT FOR
                                END IF
                        NEXT
                END IF
                CombTable(Subscript) = CombTable(Subscript) * (RootofDocFreq! / RootofAvgDocFreq!) / Divisor!

        NEXT
NEXT

ERASE Freq63

END SUB

FUNCTION ComboSum& (Bit$, Value&, Mode$, Poly!()) STATIC

REDIM Words(1 TO 15)
Multiplier! = 0

'---- find words shown by the on bits
NumbWords = 0
PolyWord! = 0
QueryLen = LEN(ExprKeys$) \ 2
FOR BitNumb = 1 TO QueryLen
        IF GetBit%(Bit$, BitNumb) THEN
```

```
                NumbWords = NumbWords + 1
                PolyWord! = PolyWord! + Poly!(BitNumb)
                Words(NumbWords) = BitNumb
        END IF
NEXT

'**** QueryLen = NumbWords '**** TEST: THY 4/8/91


'---- find all combinations of 2 words
CombSum! = 0
FOR i = 1 TO NumbWords - 1
        FOR j = i + 1 TO NumbWords

                TableNumb = (Words(i) * 15) + Words(j)        ' which combination of 2

                '-- get value from table
                PairValue! = CombTable(TableNumb)

                '** 3/20/91: no more sqr root here, see BuildCombTable for changes
                '-- compute sqr root of value and divide by Divisor
                '   with appropriate max value
                'PairValue! = SQR(PairValue!) / Divisor

                IF Mode$ = "S" THEN
                        SELECT CASE QueryLen
                                CASE 2: MaxPair! = .3
                                CASE 3: MaxPair! = 1!
                                CASE IS >= 4: MaxPair! = .9
                                CASE ELSE
                        END SELECT
                ELSE
                        SELECT CASE QueryLen
                                CASE 2: MaxPair! = .5
                                CASE 3: MaxPair! = 1.3
                                CASE 4: MaxPair! = 1.2
                                CASE IS >= 5: MaxPair! = 1.1
                                CASE ELSE
                        END SELECT
                END IF
                IF PairValue! > MaxPair! THEN PairValue! = MaxPair!
                CombSum! = CombSum! + PairValue!
        NEXT
NEXT

'-- Summed maximum values
IF Mode$ = "S" THEN
        SELECT CASE QueryLen
                CASE 2: MaxComb! = .3
                CASE 3: MaxComb! = 1.4
                CASE 4: MaxComb! = 1.8
                CASE 5: MaxComb! = 2.3
                CASE IS >= 6: MaxComb! = 2.8
                CASE ELSE: MaxComb! = 10!
        END SELECT
ELSE
        SELECT CASE QueryLen
                CASE 2: MaxComb! = .5
                CASE 3: MaxComb! = 1.6
                CASE 4: MaxComb! = 1.9
                CASE 5: MaxComb! = 2.3
                CASE IS >= 6: MaxComb! = 2.8
                CASE ELSE: MaxComb! = 10!
        END SELECT
END IF
IF CombSum! > MaxComb! THEN
        CombSum! = MaxComb!
END IF

'---- modify value using formula
IF NumbWords > 1 THEN

        ' new value = old value * formula calculated multiplier
        ' Value = sum of weights * (# of words - sum of calculated values for all pairs)

         Power! = PolyWord! - CombSum!

        IF Power! > 1 THEN
                IF Mode$ = "S" THEN
                        Multiplier! = 2 ^ Power!
                ELSE
                        Multiplier! = 1.8 ^ Power!
                END IF
        END IF
        cs# = Value& * Multiplier!
        IF cs# > 2147483647 THEN cs# = 2147483647
        ComboSum& = cs#
ELSE
        cs& = Value&
        ComboSum& = cs&
END IF

END FUNCTION

SUB DeleteWord (Expr AS ExpressionType) STATIC

GlobalStatus = EditSearch
```

```
IF Expr.SubExpr(1).Num = 0 THEN
        Chime 4
        DispMsg "ERROR: No Words to Delete. Press the Space Bar to continue:", R, c
        WaitSpace
        DispMsg "", 0, 0
        EXIT SUB
END IF

GOSUB DispPrompt

IF Lang$ <> "GERMAN" THEN
        CALL References("")
END IF

DO
        DO
                Kee$ = INKEY$
        LOOP UNTIL LEN(Kee$) > 0
        Kee = ASC(UCASE$(RIGHT$((Kee$), 1)))
        IF Kee = F10 THEN
                        IF Lang$ = "GERMAN" THEN
                                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                                IF i$ = "J" THEN
                                        GlobalStatus = NewSearch
                                        EXIT SUB
                                END IF
                        ELSE
                                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                                IF i$ = "Y" THEN
                                        GlobalStatus = NewSearch
                                        EXIT SUB
                                END IF
                        END IF
        ELSEIF Kee = ESC THEN

                EXIT DO

        ELSEIF (Kee >= ASC0 + 1 AND Kee <= ASC9) OR (Kee >= AscUpperA AND Kee <= AscUpperA + Expr.SubExpr(1).Num - 9) THEN

                n = VAL(Kee$)

                IF n = 0 THEN n = Kee - 55  'so A=10, B=11, etc.

                IF n > Expr.SubExpr(1).Num THEN
                        '-- user chose a word that doesn't exist
                        BEEP
                ELSE
                        '-- get code of chosen word
                        Code = Str2Code(Expr.SubExpr(1).Phrase, n)
                        '-- save the deleted word in a last-in-first-out stack
                        '   for use in undeleting
                        LIFO$ = LIFO$ + MKI$(Code)
                        '-- now delete the word by moving the words ahead of it
                        '   up one slot
                        ExprCodes(n).Poly = 0
                        FOR i = n TO Expr.SubExpr(1).Num - 1
                                Code2Str Expr.SubExpr(1).Phrase, i, Str2Code(Expr.SubExpr(1).Phrase, i + 1)
                                Code2Str Expr.SubExpr(1).Phrase, i + 1, 0
                                LSET ExprCodes(i) = ExprCodes(i + 1)
                        NEXT
                        '-- decrement word counter
                        Expr.SubExpr(1).Num = Expr.SubExpr(1).Num - 1
                        NumExprWords = NumExprWords - 1
                        IF Expr.SubExpr(1).Num = 0 THEN Expr.Num = 0
                        Expr.Match = -1                 '-- Reset to No full search yet
                        Expr.SubExpr(1).Match = -1      '-- Reset to No search yet
                        EXIT DO
                END IF

        ELSE
                '-- invalid keypress
                BEEP

        END IF
LOOP

DispMsg "", 0, 0

EXIT SUB

'-----------------------------------------------------

DispPrompt:

FOR i = 1 TO MinInt(9, Expr.SubExpr(1).Num)
        QPrintRC "     ", 3 + i, 4, NormAttr

        QPrintRC STR$(i), 3 + i, 6, RevAttr
NEXT

FOR i = 10 TO Expr.SubExpr(1).Num
        QPrintRC "     ", 3 + i, 4, NormAttr
        QPrintRC " " + CHR$(AscUpperA - 10 + i), 3 + i, 6, RevAttr
NEXT

IF Lang$ = "GERMAN" THEN
        DispMsg "Zahl oder Buchstabe um Wort zu loeschen    ", R, c
ELSE
```

```
        DispMsg "Enter a Number or Letter to Delete Word/Phrase or Press ESC to Cancel", R, c
END IF

RETURN

END SUB

SUB DrawBox (UlRow, UlCol, LRRow, LRCol, Frame, Col) STATIC
IF TermTypeFlag THEN
        CALL Box0(UlRow, UlCol, LRRow, LRCol, Frame, Col)
ELSE
        QPrintRC "+" + STRING$(LRCol - UlCol - 1, "-") + "+", UlRow, UlCol, Col
        FOR i = UlRow + 1 TO LRRow - 1
                QPrintRC "|", 1, UlCol, Col
                QPrintRC "|", 1, LRCol, Col
        NEXT
        QPrintRC "+" + STRING$(LRCol - UlCol - 1, "-") + "+", LRRow, UlCol, Col
END IF
END SUB

SUB FindRelatives (Expr AS ExpressionType, SubNum, Ptr, Rel() AS CollectType, NumRel, Exclude$) STATIC

SubNum = 1

DIM RelTemp AS RelativeType' temporary storage for retrieval from EMS
REDIM Collect(1 TO 1) AS CollectType
NumCollect = 0: Added = 0: NumRel = 0

ExprKeys$ = ""
NumExprKeys = 0
FOR i = 1 TO Expr.Num
        ExprKeys$ = ExprKeys$ + LEFT$(Expr.SubExpr(i).Phrase, Expr.SubExpr(i).Num * 2)
        NumExprKeys = NumExprKeys + Expr.SubExpr(i).Num
NEXT

'---- exclude list is all word from query plus parts of combkw plus parts of
'     words with prefixes.
Exclude$ = ExprKeys$ + ExcludeAdd$

END SUB

SUB FullText (First&, Last&, FileNum, Expr$)

'-----------------------------------------------------------
'                                                           '
'   FullText.bas - Nov 5, 1990   CompLex                    '
'                                                           '
' . CREATE ARRAY1 (POINTERS) FROM DICT.COD & SORT BY CODE   '
' . FIND ALL SYN. MATCHING CODES & PUT INTO ARRAY3          '
' . READ TEXT LINES FROM FILE INTO ARRAY2                  '
' . CHECK FOR SYN. MATCHES FOR ALL WORDS IN LINES          '
' . DISPLAY TEXT LINES                                     '
' . HI LITE WORDS MATCHING SYNS. IN ARRAY3                 '
'                                                           '
'-----------------------------------------------------------

LinCount = Last& - First& + 1
IF LinCount > 1000 THEN LinCount = 1000   ' maximum fitting in 128K
DIM DictTemp AS DictType                  ' var. to read ems rec. into
DictLEN = LEN(DictTemp)
DIM Array3$(1 TO 1)                       ' contains all syns.

REDIM Array1(1 TO DictCodeNum) AS Array1Type
Ems2Array Array1(1), LEN(Array1(1)), DictCodeNum, Array1EMS

'------[ find all synon. for all search words & place into array3 ]-------

FOR Counter = 1 TO (LEN(Expr$) \ 2)
        ThisCode = Str2CodeX(Expr$, Counter)

        ' binary search array1...
        L = 1
        R = DictCodeNum

        DO
                ThisElement = (CLNG(L) + R) \ 2
                IF ThisCode < Array1(ThisElement).CodeNum THEN
                        R = ThisElement - 1
                ELSE
                        L = ThisElement + 1
                END IF
        LOOP UNTIL ThisCode = Array1(ThisElement).CodeNum OR L > R

        ' if found...
        IF ThisCode = Array1(ThisElement).CodeNum THEN

                '-- backup until first match

                DO UNTIL Array1(ThisElement - 1).CodeNum <> ThisCode
                        ThisElement = ThisElement - 1
                LOOP

                '-- copy all syns. into array3$
                DO UNTIL ThisCode <> Array1(ThisElement).CodeNum
                        EmsGet1El DictTemp, LEN(DictTemp), Array1(ThisElement).RecNum, DictCodeH
                        NumOfSyns = NumOfSyns + 1
                        REDIM PRESERVE Array3$(1 TO NumOfSyns)
                        Array3$(NumOfSyns) = RTRIM$(DictTemp.Str)
```

```
                    ThisElement = ThisElement + 1
                    IF ThisElement > DictCodeNum THEN EXIT DO
                LOOP
        END IF
NEXT

ERASE Array1    ' don't need it anymore, and it's still in EMS

REDIM Text(1 TO LinCount) AS STRING * 80  ' contains text lines
REDIM Array4(1 TO LinCount, 1 TO 30)  ' contains hi-lite words info

'IF NumOfSyns = 0 THEN
'   Chine 10
'    IF Lang$ = "GERMAN" THEN
'      QPrintRC "Number of syns = 0", 1, 1, -1
'    ELSE
'      QPrintRC "Number of syns = 0", 1, 1, -1
'    END IF
'   STOP
'END IF

REDIM Array5$(1 TO NumOfSyns, 5) ' contains all syns (parsed)
REDIM Array6(1 TO NumOfSyns)     ' parallel to Array5: # of words in row

FOR Counter = 1 TO NumOfSyns

        '---- NumSynWords
        Array5$(Counter, 0) = STR$(InCount(RTRIM$(Array3$(Counter)), " ") + 1)
        Array6(Counter) = InCount(RTRIM$(Array3$(Counter)), " ") + 1

        FOR Counter1 = 1 TO VAL(Array5$(Counter, 0))
                CALL Extract(Array3$(Counter), " ", Counter1, StrtSyn, SlenSyn)
                Array5$(Counter, Counter1) = MID$(Array3$(Counter), StrtSyn, SlenSyn)
        NEXT 'counter1

NEXT 'counter

'-------------[ read text lines from file into array2 ]-------------

ThisLine = 0

FOR Counter& = First& TO Last&

        ThisLine = ThisLine + 1

        IF ThisLine > LinCount THEN EXIT FOR

        GET FileNum, Counter&, Text(ThisLine)

        Array4(ThisLine, 1) = -1     ' indicates this line was not checked

NEXT
'-------------[ display text lines & poll for avail. keys ]-------------

LinePtr = 1       ' Set line pointer
PrevLinePtr = 0
NotNewSearch:
DO

        IF LinePtr <> PrevLinePtr THEN
                PrevLinePtr = LinePtr

                ' Update the 24 lines of text
                IF AscInkee <> UP AND AscInkee <> Dn THEN

                        ' Print information bar at bottom
                IF TermTypeMode$ = "LOCAL" THEN
                        LeftChar$ = CHR$(26)
                        RightChar$ = CHR$(27)
                        ExpChar$ = "F2"
                        DirDocChar$ = "F5"
                        UpChar$ = CHR$(24)
                        DnChar$ = CHR$(25)
                        PgUPChar$ = "PgUp, "
                        PgDnChar$ = "PgDn, "
                        HomeChar$ = "HOME"
                        EndChar$ = "END"
                ELSE
                        LeftChar$ = CHR$(LeftArrowKey)
                        RightChar$ = CHR$(RightArrowKey)
                        ExpChar$ = CHR$(ShowExprKey)
                        DirDocChar$ = CHR$(DirNumKey)
                        UpChar$ = CHR$(UpArrowKey)
                        DnChar$ = CHR$(DownArrowKey)
                        PgUPChar$ = CHR$(PgUpKey) + ": PgUp, "
                        PgDnChar$ = CHR$(PgDnKey) + ": PgDn, "
                        HomeChar$ = CHR$(HomeKey)
                        EndChar$ = CHR$(EndKey)
                END IF
                IF Lang$ = "GERMAN" THEN
                        Infoline$ = ExpChar$ + ": Sucheingabe" + LeftChar$ + ": Naechstes Dok " + RightChar$ + ": Vorhergeh Dok " + Di
                ELSE
                        Infoline$ = UpChar$ + ": Up, " + DnChar$ + ": Down, " + PgUPChar$ + PgDnChar$ + HomeChar$ + ": Top, " + EndCha
                END IF
                 CALL References(Infoline$)
                END IF
                IF TermTypeMode$ = "LOCAL" THEN
                        R = 23
                ELSE
                        R = 22
```

```
END IF
FOR i = 0 TO R
        ThisLine = i + LinePtr

        IF ThisLine <= LinCount THEN
                ' print line with normal attrib.
                QPrintRC Text(ThisLine), i + 1, 1, NormAttr

                ' if this line not searched yet
                IF Array4(ThisLine, 1) < 0 THEN

                        IF Array4(ThisLine, 1) = -1 THEN
                                Array4(ThisLine, 1) = 0      ' indicates this line was checked
                                ThisWordNum = 1
                        ELSE    ' if less then -1
                                ThisWordNum = Array4(ThisLine, 1) \ -2  ' # of words already checked
                                Array4(ThisLine, 1) = 1
                        END IF

                        NumWords = InCountX(RTRIM$(Text(ThisLine)), " ") + 1

                        LineTmp$ = LCASE$(Text(ThisLine))

                        GOSUB FilterLine

                        Counter2 = ThisWordNum - 1
                        DO
                                Counter2 = Counter2 + 1

                                CALL Extract(LineTmp$, " ", Counter2, Strt, Slen)
                                CurrWord$ = MID$(LineTmp$, Strt, Slen)

' not a valid word so goto next word & inc. word count
IF LEN(CurrWord$) = 0 THEN
        IF NumWords < 80 THEN
                NumWords = NumWords + 1
        END IF
        GOTO LoopCounter2
END IF

FOR Counter3 = 1 TO NumOfSyns

        ' NumSynWords = VAL(array5$(counter3, 0))     ' test new array *** rea for spe
        CurrSynWord$ = Array5$(Counter3, 1)   ' test new array ***
        SlenSyn = LEN(CurrSynWord$)

      IF Array6(Counter3) = 1 THEN    ' if single syn. word
                IF RIGHT$(CurrSynWord$, 2) = "//" THEN
                        SlenSyn = SlenSyn - 2
                        CurrSynWord$ = LEFT$(CurrSynWord$, SlenSyn)

                        Match = (CurrSynWord$ = CurrWord$)
                ELSEIF RIGHT$(CurrSynWord$, 1) = "/" THEN
                        SlenSyn = SlenSyn - 1
                        CurrSynWord$ = LEFT$(CurrSynWord$, SlenSyn)

                        Match = (CurrWord$ = CurrSynWord$)
                ELSE
                        Match = (CurrSynWord$ = LEFT$(CurrWord$, SlenSyn))
                END IF

                IF Match THEN     ' add to array4
                        Array4(ThisLine, 1) = Array4(ThisLine, 1) + 1
                        Array4(ThisLine, (Array4(ThisLine, 1) * 2)) = Strt
                        Array4(ThisLine, (Array4(ThisLine, 1) * 2) + 1) = Slen
                END IF
        ELSE ' if combo syn.
                FirstStrt = Strt  ' indicates where to start hi-lite word group

                FOR Counter4 = 1 TO Array6(Counter3)
                        IF Counter4 > 1 THEN
                                ' get next word in line
                                ThisWordNum = Counter2 + Counter4 - 1
                                ThisLineTmp = ThisLine
                                IF ThisWordNum > NumWords THEN
                                        ThisWordNum = ThisWordNum - NumWords
                                        ThisLineTmp = ThisLine + 1
                                        IF ThisLine >= LinCount THEN
                                                Match = FALSE
                                                EXIT FOR 'counter4
                                        END IF

                                        LineTmp$ = LCASE$(Text(ThisLineTmp))
                                        GOSUB FilterLine
                                END IF

                                CALL Extract(LineTmp$, " ", ThisWordNum, Strt, Slen)
                                CurrWord$ = LCASE$(MID$(LineTmp$, Strt, Slen))
                                ' get next syn.
                                CurrSynWord$ = Array5$(Counter3, Counter4)   ' test ne
                                SlenSyn = LEN(CurrSynWord$)
                        END IF

                        IF RIGHT$(CurrSynWord$, 2) = "//" THEN
                                SlenSyn = SlenSyn - 2
                                CurrSynWord$ = LEFT$(CurrSynWord$, SlenSyn)

                                Match = (CurrWord$ = CurrSynWord$)
                        ELSEIF RIGHT$(CurrSynWord$, 1) = "/" THEN
                                SlenSyn = SlenSyn - 1
                                CurrSynWord$ = LEFT$(CurrSynWord$, SlenSyn)
```

```
                Match = (CurrWord$ = CurrSynWord$)
            ELSE
                Match = (CurrSynWord$ = LEFT$(CurrWord$, SlenSyn))
            END IF

            IF CurrSynWord$ = "a" THEN
                IF INSTR(AtList$, "/" + CurrWord$ + "/") THEN
                    Match = TRUE
                ELSE


                    Match = FALSE
                END IF
            END IF
            IF Match = FALSE THEN EXIT FOR
        NEXT 'counter4

        IF Match THEN     ' add to array4
            IF ThisLineTmp > ThisLine THEN     ' combo on 2 lines
                ' 1st line
                SlenL1 = 80 - FirstStrt
                StrtL1 = FirstStrt

                Array4(ThisLine, 1) = Array4(ThisLine, 1) + 1
                Array4(ThisLine, (Array4(ThisLine, 1) * 2)) = StrtL1
                Array4(ThisLine, (Array4(ThisLine, 1) * 2) + 1) = SlenL1

                ' 2nd line
                SlenL2 = (Strt + Slen) - 1
                StrtL2 = 1

                Array4(ThisLineTmp, 1) = ThisWordNum * -2 ' # of words
                Array4(ThisLineTmp, 2) = StrtL2
                Array4(ThisLineTmp, 3) = SlenL2
            ELSE
                Slen = (Strt + Slen) - FirstStrt
                Strt = FirstStrt

                Array4(ThisLine, 1) = Array4(ThisLine, 1) + 1
                Array4(ThisLine, (Array4(ThisLine, 1) * 2)) = Strt
                Array4(ThisLine, (Array4(ThisLine, 1) * 2) + 1) = Slen
            END IF
        END IF

        CALL Extract(LineTmp$, " ", Counter2, Strt, Slen)
        CurrWord$ = MID$(LineTmp$, Strt, Slen)

    END IF' was it a combo or sing keyword?

NEXT 'counter3
LoopCounter2:
LOOP UNTIL Counter2 = NumWords

END IF

' hi - lite words in line
IF Array4(ThisLine, 1) > 0 THEN
    FOR Counter2 = 2 TO (Array4(ThisLine, 1) * 2) STEP 2
        HLWord$ = MID$(Text(ThisLine), Array4(ThisLine, Counter2), Array4(ThisLine, Counter2 +
        QPrintRC HLWord$, i + 1, Array4(ThisLine, Counter2), RevAttr
    NEXT 'counter2
END IF
ELSE
    QPrintRC SPACE$(80), i + 1, 1, NormAttr
END IF
NEXT i
END IF

' wait for a key to be pressed
DO
    Inkee$ = INKEY$
LOOP UNTIL LEN(Inkee$)
IF LEN(Inkee$) = 1 THEN
    AscInkee = ASC(UCASE$(Inkee$))
ELSEIF LEN(Inkee$) = 2 THEN
    AscInkee = ASC(RIGHT$(Inkee$, 1)) + 200
END IF
    SELECT CASE AscInkee
        CASE UpArrowKey, UP
            IF LinePtr > 1 THEN
                LinePtr = LinePtr - 1
            END IF
        CASE DownArrowKey, Dn
            IF LinePtr < LinCount THEN
                LinePtr = LinePtr + 1
            END IF
        CASE PgUpKey, PGUP
            IF LinePtr > 1 THEN
                LinePtr = LinePtr - R
                IF LinePtr < 1 THEN
                    LinePtr = 1
                END IF
            END IF
        CASE PgDnKey, PGDN
            IF LinePtr <= LinCount - R - 1 THEN
                LinePtr = LinePtr + R
                IF LinePtr > LinCount THEN
                    LinePtr = LinCount
                END IF
```

```
                END IF
            CASE HomeKey, H
                IF LinePtr > 1 THEN
                    LinePtr = 1
                END IF
            CASE EndKey, E
                IF LinePtr < LinCount - R THEN
                    LinePtr = LinCount - R
                END IF
            CASE ESC, RightArrowKey, LeftArrowKey, DirNumKey, ShowExprKey, F2, F3, F4, F5
                ExitFlag = AscInkee
                QuitFlag = TRUE
        CASE NewSearchKey, F10
            IF Lang$ = "GERMAN" THEN
                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                IF i$ = "J" THEN
                    ExitFlag = AscInkee
                    QuitFlag = TRUE
                ELSE
                    GOTO NotNewSearch
                END IF
            ELSE
                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                IF i$ = "Y" THEN
                    ExitFlag = AscInkee
                    QuitFlag = TRUE
                ELSE
                    GOTO NotNewSearch
                END IF
            END IF
        CASE ELSE
    END SELECT
LOOP UNTIL QuitFlag

'-------------------------[ free up memory ]-----------------------

ERASE Text      ' held the document text (fixed.len string)
ERASE Array3$
ERASE Array4
ERASE Array5$
ERASE Array6

EXIT SUB

'----------------------------------------------------------------
FilterLine:

'---- replace all 's with spaces
PuncPos = 1
DO
    PuncPos = INSTR(PuncPos, LineTmp$, "'s")

    IF PuncPos > 0 THEN
        MID$(LineTmp$, PuncPos, 2) = "  "
        PuncPos = PuncPos + 2
    END IF

LOOP UNTIL PuncPos = 0

'---- replace all punctuations with spaces
FOR Fcounter = 33 TO 37
    CALL ReplaceChar(LineTmp$, CHR$(Fcounter), " ")
NEXT 'Fcounter

FOR Fcounter = 39 TO 46
    CALL ReplaceChar(LineTmp$, CHR$(Fcounter), " ")
NEXT 'Fcounter

FOR Fcounter = 58 TO 64
    CALL ReplaceChar(LineTmp$, CHR$(Fcounter), " ")
NEXT 'Fcounter

RETURN

END SUB

SUB LoadPrefixes (Prefixes$(), MeanPrefixes$(), Lang$) STATIC

IF Lang$ = "GERMAN" THEN
    RESTORE GermanPrefixes
ELSE
    RESTORE EnglishPrefixes
END IF
REDIM Prefixes$(2 TO 9)
IF Lang$ = "GERMAN" THEN
    FOR i = 2 TO 9
        READ FirstHalf$, SecondHalf$
        Prefixes$(i) = FirstHalf$ + SecondHalf$
    NEXT
ELSE
    FOR i = 2 TO 9
        READ Prefixes$(i)
    NEXT
END IF
REDIM MeanPrefixes$(3 TO 14)
IF Lang$ = "GERMAN" THEN
    FOR i = 3 TO 14
        READ FirstHalf$, SecondHalf$, ThirdHalf$
        MeanPrefixes$(i) = FirstHalf$ + SecondHalf$ + ThirdHalf$
    NEXT
```

```
ELSE
        FOR i = 3 TO 14
                READ MeanPrefixes$(i)
        NEXT
END IF
END SUB

SUB OtherWords (Expr AS ExpressionType, WordPtr) STATIC

'---- Continue finding and selecting Relatives/SWAPs until none are selected

REDIM Relatives(1 TO 1) AS CollectType

FindRelatives Expr, CurrentSub, WordPtr, Relatives(), NumRelatives, Exclude$

ShowExpr Expr

IF Expr.SubExpr(1).Num > 1 THEN
        BuildCombTable ("$") '-- indicate SWAPS is the caller
END IF

AddSwaps Expr, Exclude$, Relatives(), NumRelatives

NumSelected = SelectRelativesX(Expr, CurrentSub, Relatives(), NumRelatives)

IF GlobalStatus = NewSearch THEN EXIT SUB

'---- Display information using ShowExpr
ShowExpr Expr

END SUB

SUB PickChoice (m() AS CollectType, NumM, Label$, PickX(), NumPick, ExactFlag, Expr AS ExpressionType) STATIC

SHARED Cnf AS Config

'PickChoice allows the user to choose one or more of the keywords from
'   the list of keywords passed in M() as code#'s.  It places the choices
'   directly into the SubExpr.Phrase.

'NOTE: if there is only 1 choice passed to this routine, then it will
'      automatically return that choice as the one chosen, without
'      interacting with the user

' first set up the array of keywords from the list of code #'s
NewSearchFlag = FALSE
DO
        REDIM Word$(0 TO NumM)
        Word$(0) = Label$
        MaxWordLen = LEN(Word$(0)) + 2
        FOR i = 1 TO NumM
                Word$(i) = Dict$(m(i).Code)

        IF LEN(Word$(i)) > MaxWordLen THEN
                        MaxWordLen = LEN(Word$(i))
        END IF
NEXT
FOR i = 1 TO NumM
        SpaceLen = MaxIntX(MaxWordLen, LEN(Label$) + 2) - LEN(Word$(i)) + 2
        Word$(i) = Word$(i) + SPACE$(SpaceLen)
NEXT
IF NumM = 1 AND (GlobalStatus = NewSearch OR GlobalStatus = AddWords) AND ExactFlag = TRUE THEN
' only one choice (during a Term Search or a Narrow Search)
' so return it automatically without a menu or a keypress
        NumPick = 1
        PickX(1) = 1
        CALL Chime(6)
ELSE
        ' push the window location to the lower right
        IF TermTypeMode$ = "LOCAL" THEN
                LOCATE 24 - MinIntX(NumM, 12) - 3, 80 - MaxWordLen - 7, 0
        ELSE
                LOCATE 24 - MinIntX(NumM, 12) - 4, 80 - MaxWordLen - 7, 0
        END IF
        ' call PickList to get choices (total maximum of 15 allowed in SubExpression)
        IF TermTypeMode$ = "LOCAL" THEN
                IF Lang$ = "GERMAN" THEN
                        CALL References("DANACH ENTER TASTE    " + CHR$(24) + CHR$(25) + " PGUP,PGDN")
                ELSE
                        CALL References(CHR$(24) + ": UP, " + CHR$(25) + ": DOWN, PGUP, PGDN, SPACE BAR: SELECT, ENTER: DONE,
                END IF
        ELSE
                IF Lang$ = "GERMAN" THEN
                        CALL References("DANACH ENTER TASTE    " + CHR$(24) + CHR$(25) + " PGUP,PGDN")
                ELSE
                        CALL References("U: UP, X: DOWN, E: PGUP, C: PGDN, SPACE BAR: SELECT, ENTER: DONE, ESC: CANCEL")
                END IF
        END IF
        CALL PickList(Word$(), PickX(), NumPick, Cnf, Expr, NewSearchKey, ShowExprKey, HomeKey, EndKey, PgUpKey, PgDnKey, UpAr
        IF NumPick = -10 THEN 'NewSearch
                IF Lang$ = "GERMAN" THEN
                        i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                        IF i$ = "J" THEN NewSearchFlag = TRUE
                ELSE
                        i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                        IF i$ = "Y" THEN NewSearchFlag = TRUE
                END IF
        END IF
END IF
LOOP UNTIL NumPick <> -10 OR NewSearchFlag
```

```
ERASE Word$

LOCATE , , 0

END SUB

SUB RankRecords (Recs() AS RecInfoType, NumFoundZ, Expr AS ExpressionType) STATIC

DIM AvgRankTot AS AvgRankType
DIM KYInfo AS KeyInfoLONG
KYInfoLen = LEN(KYInfo)
MaxWidth = INT(SQR(NumRecords&)) + 1
REDIM Record(1 TO MaxWidth, 1 TO MaxWidth) AS STRING * 1
'-- allocate an EMS array for the document ranking
DIM RankTot AS RankInfo, El1 AS RankInfo, El2 AS RankInfo
DIM BlankRankTot AS RankInfo
RankTotLEN = LEN(RankTot)
NumPages = NumRecords& * RankTotLEN \ SixteenK + 1
EmsAlloc NumPages, RankTotEMS, "Ranking Totals EMS Storage"

'---- allocate an EMS array for the summed values
DIM Value AS BitValueLong
DIM BlankValue AS BitValueLong
ValueLEN = LEN(Value)
NumPages = NumRecords& * ValueLEN \ SixteenK + 1
EmsAlloc NumPages, RankEMS, "Ranking EMS Storage"

'-- clear (to 0) the EMS memory we allocated
EMSPF = EmsGetPFSeg%
FOR i = 1 TO NumPages
       CALL EmsMapMem(RankEMS, 1, i)    ' map logical page i to physical page 0
        CALL InitMem(EMSPF, 0, 8192, 0)  ' clear physical page of memory to 0
NEXT

IF Lang$ = "GERMAN" THEN
       DispMsg "GEWICHTUNG DER GEFUNDENEN DOKUMENTE .....", 0, 0
ELSE
       DispMsg "Ranking Documents .....", 0, 0
END IF

ExprKeys$ = ""
QueryNum = 0
FOR i = 1 TO Expr.Num
       ExprKeys$ = ExprKeys$ + LEFT$(Expr.SubExpr(i).Phrase, Expr.SubExpr(i).Num * 2)
       QueryNum = QueryNum + Expr.SubExpr(i).Num
NEXT

BuildCombTable ("R") '-- indicate RANKing is the caller

QueryLen = LEN(ExprKeys$) \ 2
REDIM Poly!(1 TO QueryLen)
FOR i = 1 TO QueryLen
        Poly!(i) = (ExprCodes(i).Poly / PolyAvg!) ^ .25
NEXT

NumRank& = 0: EmptyFlag = FALSE

FullSearch:

FOR i = 1 TO Expr.Num

       FOR j = 1 TO Expr.SubExpr(i).Num
              MemFlag = FALSE
              Code = Str2Code(Expr.SubExpr(i).Phrase, j)
              EmsGet PolyValue, PolyLEN, CLNG(Code), PolySegEMS '** TMY 5/13/91
              FreeSpace& = FRE(-1)
              'if there is enough memory then read the whole block
              MemReq& = CLNG(KYIndx(Code).Num) * KYInfoLen
              IF FreeSpace& > MemReq& + 1024 THEN  'Leave 1K free
                     REDIM KYInfoArray(1 TO KYIndx(Code).Num) AS KeyInfoLONG
                     'FGetA can read only 64K, so if we have more, read it in two steps
                     IF MemReq& < ThirtyTwoK& * 2 THEN
                            FSeek KYInvertDatFILE, (KYIndx(Code).Index - 1) * KYInfoLen
                            FGetA2 KYInvertDatFILE, SEG KYInfoArray(1), CLNG(KYIndx(Code).Num) * KYInfoLen
                     ELSE
                            FSeek KYInvertDatFILE, (KYIndx(Code).Index - 1) * KYInfoLen
                            FGetA2 KYInvertDatFILE, SEG KYInfoArray(1), CLNG(KYIndx(Code).Num \ 2) * KYInfoLen
                            FSeek KYInvertDatFILE, (KYIndx(Code).Index - 1) * KYInfoLen + CLNG(KYIndx(Code).Num \ 2) * KYInfoLen
                            FGetA2 KYInvertDatFILE, SEG KYInfoArray(KYIndx(Code).Num \ 2 + 1), CLNG(KYIndx(Code).Num \ 2 + KYIndx(
                     END IF
                     MemFlag = TRUE
              END IF
              FOR Index& = 1 TO KYIndx(Code).Num
                     IF MemFlag THEN
                            LSET KYInfo = KYInfoArray(Index&)
                     ELSE
                            FGetRT KYInvertDatFILE, KYInfo, KYIndx(Code).Index - Index& - 1, KYInfoLen
                     END IF
                     Horiz = KYInfo.Rec MOD MaxWidth
                     Vert = INT(KYInfo.Rec / MaxWidth) + 1
                     IF Horiz = 0 THEN Horiz = MaxWidth: Vert = Vert - 1

                     '---- give a boost to the original words in expression
                     IF KeyInstr%(OriginalExpr$, MKI$(Code)) THEN
                            KYInfo.Value = KYInfo.Value * Boost!
                     END IF

                     '-- clear Value variable
                     Value = BlankValue
```

```
                        '— check against first appearing only for non rare words
                        IF QueryNum > 3 AND KYIndx(Code).Num > Limit * 5 AND NOT EmptyFlag THEN
                                'to pick up documents even there is only one keyword(for small set of documents)
                                'change this block to GOSUB KeepRec
'vvvv
```

```
IF AvgRankFILE < 0 THEN
        CALL FCreate("AvgRank")
        FOpenAll "AvgRank", 2, 4, AvgRankFILE
        AvgRankTot.Num = 0
        AvgRankTot.Value = 0
        FOR i& = 1 TO 17
                CALL FPutRT(AvgRankFILE, AvgRankTot, i&, LEN(AvgRankTot))
        NEXT i&
END IF

FGetRT AvgRankFILE, AvgRankTot, CLNG(Expr.SubExpr(1).Num), LEN(AvgRankTot)

IF AvgRankTot.Num = 0 THEN
        AvgRankTot.Value = MatchRecVals(1).Value
        NewAvgFirstVal! = MatchRecVals(1).Value
ELSE
        IF AvgRankTot.Num = 1 THEN
                NewAvgFirstVal! = (AvgRankTot.Value * .5) + (MatchRecVals(1).Value * .5)
        ELSE
                NumDivNumPlus1! = AvgRankTot.Num / (AvgRankTot.Num + 1)
                NewAvgFirstVal! = (AvgRankTot.Value * NumDivNumPlus1!) + (MatchRecVals(1).Value / (AvgRankTot.Num + 1))
        END IF
END IF

AvgFirstVal! = AvgRankTot.Value
AvgRankTot.Value = NewAvgFirstVal!
AvgRankTot.Num = AvgRankTot.Num + 1

FPutRT AvgRankFILE, AvgRankTot, CLNG(Expr.SubExpr(1).Num), LEN(AvgRankTot)

FClose AvgRankFILE

EmsRelMem RankTotEMS
ERASE Record

DispMsg "", 0, 0

EXIT SUB

'=============================================================

KeepRec:

IF Record(Horiz, Vert) < "2" THEN
        '-- this is the second time we've seen this document, so add it to
        '   our list of found documents to be ranked
        NumRank& = NumRank& + 1
        RankTot = BlankRankTot  '-- clear the variable
        RankTot.Rec = KYInfo.Rec
        EmsSet RankTot, RankTotLEN, NumRank&, RankTotEMS
END IF

EmsGet Value, ValueLEN, KYInfo.Rec, RankEMS
Value.Value = Value.Value + KYInfo.Value * PolyValue.Value
SetBit Value.Bit, j, 1
EmsSet Value, ValueLEN, KYInfo.Rec, RankEMS
Record(Horiz, Vert) = "2"

RETURN

END SUB

SUB ReWriteHist (Curr)

'-- move the solid bar
IF TermTypeMode$ = "LOCAL" THEN
        HistStart = 25
ELSE
        HistStart = 24
END IF

FullFlag = FALSE

Ratio! = (MatchRecVals(1).Value ^ .5) / (AvgFirstVal! ^ .5)

HistFlag = FALSE
Start = 1
IF Curr = 1 THEN
        St = 1
ELSE
        St = Curr - 1
END IF
IF Curr = 25 THEN
        Fin = 25
ELSE
        Fin = Curr + 1
END IF
IF Fin > UBOUND(MatchRecVals) THEN Fin = UBOUND(MatchRecVals)
FOR i = St TO Fin 'NumBars
        FOR j = 1 TO 16
                IF (MatchRecVals(i).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1), NormAttr
                        QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> 1 THEN QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1) - 2, 0

                        IF j = 1 AND i <> Curr THEN
                                QPrintRC "   ", HistStart - 1, 3 * (i - Start + 1) - 2, RevAttr
                                QPrintRC STR$(i), HistStart - 1, 3 * (i - Start + 1) - 2, RevAttr
                                IF i <> 1 THEN
                                        QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, 0
                                ELSE
```

```
                        QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, (BG AND 1)
                    END IF
                    QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                    QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                    IF i <> 1 THEN QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                END IF

                IF i = Curr THEN
                  QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1), NormAttr
                  QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1) - 1, NormAttr
                  IF i <> 1 THEN QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1) - 2, 0
                  IF j = 1 THEN
                        QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                        QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> 1 THEN QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                  END IF
                END IF
            END IF
              'if no bar, print number
              IF (SQR(MatchRecVals(i).Value) < (.0625 * SQR(MatchRecVals(1).Value))) THEN
                    IF i = Curr THEN
                        QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1), NormAttr
                        QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> 1 THEN
                            QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, 0
                        ELSE
                            QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, NormAttr
                        END IF
                        QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                        QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> 1 THEN
                            QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, 0
                        ELSE
                            QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, NormAttr
                        END IF

                    ELSE
                        QPrintRC "   ", HistStart - 1, 0 + 3 * (i - Start + 1) - 2, RevAttr
                        QPrintRC STR$(i), HistStart - 1, 0 + 3 * (i - Start + 1) - 2, RevAttr
                        IF i <> 1 THEN
                            QPrintRC HighChar$, HistStart - 1, 0 + 3 * (i - Start + 1) - 2, 0
                        ELSE
                            QPrintRC HighChar$, HistStart - 1, 0 + 3 * (i - Start + 1) - 2, NormAttr
                        END IF
                        QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                        QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> 1 THEN QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                    END IF
                END IF
        NEXT j
NEXT i
IF Ratio! < 1! THEN
        FMsgRow = 24 - 16
ELSE
        FMsgRow = 24 - 16! / Ratio!
END IF
FMsgCol = 1

IF TermTypeFlag THEN
        ch = 196 '-
ELSE
        ch = ASC("-")
END IF

IF Lang$ = "GERMAN" THEN
        QPrintRC STRING$(11, ch) + "HISTORISCHER DURCHSCHNITTSWERT FUER ERSTE DOKUMENTE" + STRING$(14, ch), FMsgRow, FMsgCol, NormAttr
ELSE
        QPrintRC STRING$(11, ch) + " AVERAGE RELEVANCE OF FIRST DOCUMENT FOR SIMILAR QUERIES" + STRING$(10, ch), FMsgRow, FMsgCol, Nor
END IF

IF Curr > 1 THEN

        IF MatchRecVals(Curr).Value < .6 * MatchRecVals(Curr - 1).Value THEN
          IF Lang$ = "GERMAN" THEN
                QPrintRC "Dokument" + STR$(Curr) + " vermutlich weniger relevant als vorhergehendes", 7, 14, -1
          ELSE
                QPrintRC "Document" + STR$(Curr) + " may be less relevant than previous document ", 7, 15, -1
  END IF
  FullFlag = TRUE
  END IF
END IF
IF Curr > 2 THEN
        IF MatchRecVals(Curr).Value < .4 * MatchRecVals(Curr - 2).Value THEN
         IF Lang$ = "GERMAN" THEN
                QPrintRC "Dokument" + STR$(Curr) + " vermutlich weniger relevant als vorhergehendes ", 7, 14, -1
         ELSE
                QPrintRC "Document" + STR$(Curr) + " may be less relevant than previous documents", 7, 15, -1
         END IF
        FullFlag = TRUE
        END IF
END IF

CALL HistMessage

DO
        DO
          ch$ = INKEY$
        LOOP UNTIL ch$ <> ""
        IF LEN(ch$) = 1 THEN
                c = ASC(UCASE$(ch$))
```

```
        ELSEIF LEN(ch$) = 2 THEN
                c = ASC(RIGHT$(ch$, 1)) + 200
        END IF
        SELECT CASE c
        CASE ESC, CR, RightArrowKey, LeftArrowKey, DirNumKey, ShowExprKey, HomeKey, EndKey, 275, 277, F3, F4, F5, F2, HM, EN
                HistFlag = c
        CASE NewSearchKey, F10
        IF Lang$ = "GERMAN" THEN
                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                IF i$ = "J" THEN
                        HistFlag = c
                END IF
         ELSE
                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                IF i$ = "Y" THEN
                 HistFlag = c
                END IF
         END IF
        END SELECT
LOOP UNTIL HistFlag

END SUB

SUB ScrollHist (Curr, Direction$, MaxNum)
IF TermTypeMode$ = "LOCAL" THEN
        HistStart = 25
ELSE
        HistStart = 24
END IF
IF (Curr <= MaxNum AND NOT LastFlag) OR (Direction$ = "R") THEN
  IF Direction$ = "L" THEN
        ScrollL 7, 2, HistStart - 1, 80, 3, -1
QPrintRC "   ", 9, 62, NormAttr
QPrintRC "   ", 10, 62, NormAttr
FOR j = 1 TO 16
          ' QPrintRC " ", 24 - j, 1, 21
                IF (MatchRecVals(Curr - 1).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HistChar$, HistStart - j, 72, NormAttr
                        QPrintRC HistChar$, HistStart - j, 71, NormAttr
                        QPrintRC HistChar$, HistStart - j, 70, 0
                END IF
         NEXT
         FOR j = 1 TO 16
                        IF (MatchRecVals(Curr).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                          QPrintRC HighChar$, HistStart - j, 75, NormAttr
                          QPrintRC HighChar$, HistStart - j, 74, NormAttr
                          QPrintRC HighChar$, HistStart - j, 73, 0
                        END IF
         NEXT
         'if no bar then print just number
         IF (MatchRecVals(Curr).Value ^ .5 < (.0625 * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HighChar$, HistStart - 1, 75, NormAttr
                        QPrintRC HighChar$, HistStart - 1, 74, NormAttr
                        QPrintRC HighChar$, HistStart - 1, 73, 0
                        QPrintRC HighChar$, HistStart - 2, 75, NormAttr
                        QPrintRC HighChar$, HistStart - 2, 74, NormAttr
                        QPrintRC HighChar$, HistStart - 2, 73, 0
         END IF
         IF (MatchRecVals(Curr).Value ^ .5 < (.0625 * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HistChar$, HistStart - 1, 72, NormAttr
                        QPrintRC HistChar$, HistStart - 1, 71, NormAttr
                        QPrintRC HistChar$, HistStart - 1, 70, 0
                        QPrintRC HistChar$, HistStart - 2, 72, NormAttr
                        QPrintRC HistChar$, HistStart - 2, 71, NormAttr
                        QPrintRC HistChar$, HistStart - 2, 70, 0
         END IF
         QPrintRC STR$(Curr - 1), HistStart - 1, 70, RevAttr
         QPrintRC " ", HistStart - 1, 70, 0      'NormAttr
         IF Curr = MaxNum THEN
                LastFlag = TRUE
         ELSE
                LastFlag = FALSE
         END IF
 ELSE 'Direction$="R"
         LastFlag = FALSE
         ScrollR 7, 2, HistStart - 1, 80, 3, -1
         CALL ClearScr0(7, 76, HistStart - 1, 80, NormAttr)' clear right portion
         FOR j = 1 TO 16
              'QPrintRC " ", HistStart - j, 1, 21
              IF (MatchRecVals(Curr - HistStart).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HistChar$, HistStart - j, 3, NormAttr
                        QPrintRC HistChar$, HistStart - j, 2, NormAttr
     QPrintRC HistChar$, HistStart - j, 1, 0
              END IF
         NEXT
         FOR j = 1 TO 16
              IF (MatchRecVals(Curr - 23).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                        QPrintRC HistChar$, HistStart - j, 4, 0
              END IF
         NEXT

         FOR j = 1 TO 16
                        IF (MatchRecVals(Curr).Value ^ .5 >= (.0625 * j * MatchRecVals(1).Value ^ .5)) THEN
                          QPrintRC HighChar$, HistStart - j, 75, NormAttr
                          QPrintRC HighChar$, HistStart - j, 74, NormAttr
                          QPrintRC HighChar$, HistStart - j, 73, 0
                        END IF
         NEXT
                        QPrintRC "  ", HistStart - 1, 2, RevAttr
                        QPrintRC LTRIM$(STR$(Curr - HistStart)), HistStart - 1, 2, RevAttr
```

```
QPrintRC " ", HistStart-1, 1, 0      'RevAttr
 IF (MatchRecVals(Curr).Value ^ .5 < (.0625 * MatchRecVals(1).Value ^ .5)) THEN
                QPrintRC HighChar$, HistStart - 1, 75, NormAttr
                QPrintRC HighChar$, HistStart - 1, 74, NormAttr
                QPrintRC HighChar$, HistStart - 1, 73, 0
                QPrintRC HighChar$, HistStart - 2, 75, NormAttr
                QPrintRC HighChar$, HistStart - 2, 74, NormAttr
                QPrintRC HighChar$, HistStart - 2, 73, 0
 END IF
 IF (MatchRecVals(Curr - HistStart).Value ^ .5 < (.0625 * MatchRecVals(1).Value ^ .5)) THEN
                QPrintRC HistChar$, HistStart - 2, 3, NormAttr
                QPrintRC HistChar$, HistStart - 2, 2, NormAttr
                                QPrintRC HistChar$, HistStart - 2, 1, 0
                                QPrintRC STR$(Curr - HistStart), HistStart - 1, 1, RevAttr
                                QPrintRC " ", HistStart, 1, 0      'RevAttr
                END IF

        END IF


'   IF Lang$ = "GERMAN" THEN
'      QPrintRC STRING$(10, 32) + " RELEVANZ GRAPHIK (WEISSER BALKEN IST GEGENWAERTIGES DOKUMENT)" + STRING$(10, 32), 7, 1, -1
'   ELSE
'      QPrintRC STRING$(10, 32) + "  RELATIVE DOCUMENT RELEVANCE (SOLID BAR IS CURRENT DOCUMENT)" + STRING$(10, 32), 7, 1, -1
'   END IF
        IF Ratio! < 1! THEN
                FMsgRow = 24 - 16
        ELSE
                FMsgRow = 24 - 16! / Ratio!
        END IF
        FMsgCol = 1
        IF TeraTypeFlag THEN
                ch = 196
        ELSE
                ch = ASC("-")
  END IF
        IF Lang$ = "GERMAN" THEN
                QPrintRC STRING$(12, ch) + " HISTORISCHER DURCHSCHNITTSWERT FUER ERSTE DOKUMENTE " + STRING$(12, ch), FMsgRow, FMsgCol
        ELSE
                QPrintRC STRING$(10, ch) + " AVERAGE RELEVANCE OF FIRST DOCUMENT FOR SIMILAR QUERIES " + STRING$(10, ch), FMsgRow, FMs
        END IF
END IF
CALL HistMessage
DO
        DO
          ch$ = INKEY$
        LOOP UNTIL ch$ <> ""
        IF LEN(ch$) = 1 THEN
                c = ASC(UCASE$(ch$))
        ELSEIF LEN(ch$) = 2 THEN
                c = ASC(RIGHT$(ch$, 1)) + 200
        END IF
        SELECT CASE c
        CASE ESC, CR, RightArrowKey, LeftArrowKey, DirNumKey, ShowExprKey, HomeKey, EndKey, 275, 277, F3, F4, F5, F2, HM, EN
                HistFlag = c
        CASE NewSearchKey, F10
        IF Lang$ = "GERMAN" THEN
                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                IF i$ = "J" THEN
                        HistFlag = c
                END IF
         ELSE
                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                IF i$ = "Y" THEN
                 HistFlag = c
                END IF
         END IF
        END SELECT
LOOP UNTIL HistFlag

END SUB

FUNCTION SelectRelatives% (Expr AS ExpressionType, CurrentSub, TopKeys() AS CollectType, NumTopKeys) STATIC

'-- current subexpression is always 1
CurrentSub = 1

'---- we're now editing the current SubExpression
Expr.Num = CurrentSub

'---- no full search has been done yet for this modified expression
Expr.Match = -1
CALL ShowExpr(Expr)

LOCATE 2, 30
IF Lang$ = "GERMAN" THEN
  Label$ = "LEERTASTE fuer ein oder mehrere Worte"
ELSE
  Label$ = "Select One or More Words"
END IF

IF Expr.SubExpr(1).Num = 15 THEN EXIT FUNCTION
REDIM PickedX(1 TO 15 - Expr.SubExpr(1).Num)

CALL PickChoice(TopKeys(), NumTopKeys, Label$, Picked(), NumPicked, ExactFlag, Expr)

IF NumPicked = -10 THEN ' F10 pressed

        GlobalStatus = NewSearch
```

```
ELSEIF NumPicked > 0 THEN

        '—— store the selected words directly in the current subexpression

        '— add number of words selected to number of words in expression
        NumExprWords = NumExprWords + NumPicked

        REDIM PRESERVE ExprCodes(1 TO NumExprWords) AS CodePolyType

        FOR i = NumExprWords - NumPicked + 1 TO NumExprWords
                ExprCodes(i).Code = TopKeys(Picked(i - NumExprWords + NumPicked)).Code
                EmsGet1El ExprCodes(i).Poly, LEN(ExprCodes(1).Poly), ExprCodes(i).Code, PolySeayEMS
'?      IF INSTR((RIGHT$(x$, LEN(x$) - 5)), " ") <> 0 THEN
'?         ExprCodes(NumExprWords).Poly = ExprCodes(NumExprWords).Poly * 1.4
'?      END IF
        NEXT

        SortT ExprCodes(1), NumExprWords, Descend, LEN(ExprCodes(1)), 2, -3

        Expr.SubExpr(CurrentSub).Num = NumExprWords

        FOR k = 1 TO NumExprWords
                Code2Str Expr.SubExpr(CurrentSub).Phrase, k, ExprCodes(k).Code
        NEXT

        '— set the status to adding Swaps
        GlobalStatus = SWAPS

END IF

'— return TRUE if words were selected
SelectRelativesX = NumPicked > 0

END FUNCTION

SUB ShowHist (NumShow, Curr, Expr AS ExpressionType) STATIC

'— Shows the histogram
FullFlag = FALSE
IF TermTypeMode$ = "LOCAL" THEN
        IF Lang$ = "GERMAN" THEN
                Infoline$ = CHR$(26) + ":Naesch Dok   " + CHR$(27) + ":Vorher Dok   ESC:Menue  ENTER:Kurzfassung "
        ELSE
                Infoline$ = CHR$(27) + ": Prev, " + CHR$(26) + ": Next, HOME: First, END: Last, ENTER: Highlights, ESC: Go Back"
        END IF
        HistStart = 25
ELSE
        IF Lang$ = "GERMAN" THEN
                Infoline$ = CHR$(LeftKey) + ": Prev,  " + CHR$(RightKey) + ": Next, " + CHR$(HomeKey) + ": First, " + CHR$(EndKey) + "
        ELSE
                Infoline$ = CHR$(LeftArrowKey) + ": Prev, " + CHR$(RightArrowKey) + ": Next, " + CHR$(HomeKey) + ": First, " + CHR$(En
        END IF
        HistStart = 24
END IF
CALL References(Infoline$)
Ratio! = SQR(MatchRecVals(1).Value) / SQR(AvgFirstVal!)

HistFlag = FALSE

REDIM AScr2X(1 TO 2000)
CALL ClearScr0(7, 1, HistStart - 1, 80, NormAttr)' clear bottom portion

'IF Lang$ = "GERMAN" THEN
'   RelevanceStr$ = "RELEVANZ GRAPHIK (WEISSER BALKEN IST GEGENWAERTIGES DOKUMENT)"
'ELSE
'   RelevanceStr$ = "RELATIVE DOCUMENT RELEVANCE (SOLID BAR IS CURRENT DOCUMENT)"
'END IF
'RelStrLen = LEN(RelevanceStr$)
'SpcLeft = (80 - RelStrLen) \ 2
'SpcRight = 80 - RelStrLen - SpcLeft
'RelevanceStr$ = SPACE$(SpcLeft) + RelevanceStr$ + SPACE$(SpcRight)
'QPrintRC RelevanceStr$, 24, 1, RevAttr

IF Curr <= 25 THEN
        Finish = MinInt%(NumShow, 25)
        Start = 1
ELSE
        Finish = Curr
        Start = Finish - 24
END IF

FOR i = Start TO Finish  'NumBars
        FOR j = 1 TO 76
                IF (SQR(MatchRecVals(i).Value) >= (.0625 * j * SQR(MatchRecVals(1).Value))) THEN
                        QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1), NormAttr
                        QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1) - 1, NormAttr
                        IF i <> Start THEN QPrintRC HistChar$, HistStart - j, 3 * (i - Start + 1) - 2, 0

                        IF j = 1 AND i <> Curr THEN
                                QPrintRC "   ", HistStart - 1, 3 * (i - Start + 1) - 2, RevAttr
                                QPrintRC STR$(i), HistStart - 1, 3 * (i - Start + 1) - 2, RevAttr
                                IF i <> Start THEN
                                        QPrintRC " ", HistStart - 1, 3 * (i - Start + 1) - 2, 0
                                ELSE
                                        QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, 1
                                END IF
                                QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                                QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                                IF i <> Start THEN QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                        END IF
```

```
                    IF i = Curr THEN
                      QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1), NormAttr
                      QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1) - 1, NormAttr
                      IF i <> Start THEN QPrintRC HighChar$, HistStart - j, 3 * (i - Start + 1) - 2, 0
                      IF j = 1 THEN
                              QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                              QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                              IF i <> Start THEN QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                      END IF
                    END IF
                END IF
                  'if no bar, print number
                  IF (SQR(MatchRecVals(i).Value) < (.0625 * SQR(MatchRecVals(1).Value))) THEN
                          IF i = Curr THEN
                                  QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1), NormAttr
                                  QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 1, NormAttr
                                  IF i <> Start THEN QPrintRC HighChar$, HistStart - 1, 3 * (i - Start + 1) - 2, 0
                                  QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                                  QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                                  IF i <> Start THEN QPrintRC HighChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                                ELSE
                                        QPrintRC "   ", HistStart - 1, 0 + 3 * (i - Start + 1) - 2, RevAttr
                                        QPrintRC STR$(i), HistStart - 1, 0 + 3 * (i - Start + 1) - 2, RevAttr
                                        IF i <> Start THEN QPrintRC HighChar$, HistStart - 1, 0 + 3 * (i - Start + 1) - 2, 0
                                        QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1), NormAttr
                                        QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 1, NormAttr
                                        IF i <> Start THEN QPrintRC HistChar$, HistStart - 2, 3 * (i - Start + 1) - 2, 0
                                END IF
                  END IF
        NEXT j
NEXT i
IF Ratio! < 1! THEN
        FMsgRow = 24 - 16
ELSE
        FMsgRow = 24 - 16! / Ratio!
END IF
FMsgCol = 1
IF TeraTypeFlag THEN
        ch = 196
ELSE
        ch = ASC("-")
END IF

IF Lang$ = "GERMAN" THEN
        QPrintRC STRING$(12, ch) + " HISTORISCHER DURCHSCHNITTSWERT FUER ERSTE DOKUMENTE " + STRING$(12, ch), FMsgRow, FMsgCol, NormAt
ELSE
        QPrintRC STRING$(10, ch) + " AVERAGE RELEVANCE OF FIRST DOCUMENT FOR SIMILAR QUERIES " + STRING$(10, ch), FMsgRow, FMsgCol, No
END IF

IF Curr > 1 THEN
        IF MatchRecVals(Curr).Value < .6 * MatchRecVals(Curr - 1).Value THEN
          IF Lang$ = "GERMAN" THEN
                QPrintRC "Dokument" + STR$(Curr) + " vermutlich weniger relevant als vorhergehendes", 7, 14, -1
          ELSE
                QPrintRC "Document" + STR$(Curr) + " may be less relevant than previous document", 7, 15, -1
          END IF
          FullFlag = TRUE
  END IF
END IF
IF Curr > 2 THEN
        IF MatchRecVals(Curr).Value < .4 * MatchRecVals(Curr - 2).Value THEN
         IF Lang$ = "GERMAN" THEN
                QPrintRC "Dokument" + STR$(Curr) + " vermutlich weniger relevant als vorhergehendes", 7, 14, -1
         ELSE
                QPrintRC "Document" + STR$(Curr) + " may be less relevant than previous documents", 7, 15, -1
         END IF
         FullFlag = TRUE
        END IF
END IF
CALL HistMessage
DO
        DO
          ch$ = INKEY$
        LOOP UNTIL ch$ <> ""
        IF LEN(ch$) = 1 THEN
                c = ASC(UCASE$(ch$))
        ELSEIF LEN(ch$) = 2 THEN
                c = ASC(RIGHT$(ch$, 1)) + 200
        END IF
        SELECT CASE c
        CASE ESC, CR, RightArrowKey, LeftArrowKey, DirNumKey, ShowExprKey, HomeKey, EndKey, 275, 277, F3, F4, F5, F2, HM, ER
                HistFlag = c
        CASE NewSearchKey, F10
        IF Lang$ = "GERMAN" THEN
                i$ = Question$("NEUE SUCHE ? [J/N]  ", "JN", "ACHTUNG !")
                IF i$ = "J" THEN
                        HistFlag = c
                END IF
         ELSE
                i$ = Question$("New Search? [Y/N]", "YN", "WARNING!")
                IF i$ = "Y" THEN
                 HistFlag = c
                END IF
          END IF
         END SELECT
LOOP UNTIL HistFlag

'CALL MScrnRest(1, 1, 25, 80, SEG AScr2X(1))
ERASE AScr2X

END SUB
```

```
SUB SortEMSRankInfo (HandleX, NumEls&) STATIC

'-- sorts RankInfo types by Value in Descending order

DIM Array1 AS RankInfo, Array2 AS RankInfo
ArrayLEN = LEN(Array1)

Span& = NumEls& \ 2
DO WHILE Span& > 0
        FOR i& = Span& TO NumEls& - 1
                j& = i& - Span& + 1
                FOR j& = (i& - Span& + 1) TO 1 STEP -Span&
                        EmsGet Array1, ArrayLEN, j&, HandleX
                        EmsGet Array2, ArrayLEN, j& + Span&, HandleX

                        IF Array2.Value <= Array1.Value THEN EXIT FOR

                        '-- Swap array elements that are out of order.

                        EmsSet Array1, ArrayLEN, j& + Span&, HandleX
                        EmsSet Array2, ArrayLEN, j&, HandleX
                NEXT j&
        NEXT i&
        Span& = Span& \ 2
LOOP

END SUB

SUB SortSwapEMS (HandleX, NumElsX) STATIC
'-- sorts Synth BitValue types by Value in Descending order

DIM Array1 AS BitValue, Array2 AS BitValue
ArrayLEN = LEN(Array1)

Span& = NumEls \ 2
DO WHILE Span& > 0
        FOR i& = Span& TO NumEls - 1
                j& = i& - Span& + 1
                FOR j& = (i& - Span& + 1) TO 1 STEP -Span&
                        EmsGet Array1, ArrayLEN, j&, HandleX
                        EmsGet Array2, ArrayLEN, j& + Span&, HandleX

                        IF Array2.Value <= Array1.Value THEN EXIT FOR

                        '-- Swap array elements that are out of order.

                        EmsSet Array1, ArrayLEN, j& + Span&, HandleX
                        EmsSet Array2, ArrayLEN, j&, HandleX
                NEXT j&
        NEXT i&
        Span& = Span& \ 2
LOOP

END SUB

SUB TermMatch (Term$, Expr AS ExpressionType, a() AS CollectType, NumM, Repeate, ExactFlag) STATIC

' collect code #'s from the Dictionary which match Term$
' uses the same technique as the CheckKey in the AIM program
' which uses the table of the first 3 chars and does a
' "reverse" match, i.e., matching all words in the dictionary
' to the Term$

' Note that the arrays necessary for these routines are globally
' shared, so do not need to be passed

' Also Note: The SINGKEY.STR and COMBKEY.STR used below are not
' in memory, but are accessed directly from disk using FGetRT

DIM WordCompare AS SingKeyType
DIM CombKeyTemp AS ComboKeyType
DIM SingKeyTemp AS SingKeyType
DIM DictTemp AS DictType
Repeate = FALSE
IF Lang$ = "ENGLISH" THEN
        Term$ = Term$
ELSE
        Term$ = LCASE$(Term$)
END IF

ExactFlag = FALSE
' replace only SOME punctuation with spaces
New$ = " "
Old$ = ":/.-()[]{}"
FOR j = 1 TO LEN(Old$)
        CALL ReplaceChar(Term$, MID$(Old$, j, 1), New$)
NEXT

NumM = 0
SELECT CASE ASC(Term$)
  CASE AscA TO ASCZ: SearchStep = 1
  CASE AscUpperA TO ASCUpperZ:
                        IF ASC(MID$(Term$, 2, 1)) < AscUpperA OR ASC(MID$(Term$, 2, 1)) > ASCUpperZ THEN
                      SearchStep = 2
                        END IF
  CASE ELSE:          SearchStep = 3
END SELECT
IF INSTR(Term$, " ") <> 0 GOTO FindCombKey
OriginTerm$ = Term$
' check if the first 3 letters of the word return
' a valid range from the 3-dimensional table array
```

```
IF FirstLastX(LCASE$(Term$), First, Last, Sing) THEN 'yes, search thru range
AddSearchAgain:
        FOR j = First TO Last
                CALL FGetRT(SingKeyFILE, SingKeyTemp, CLNG(j), LEN(SingKeyTemp))
                CurrKey$ = RTRIM$(SingKeyTemp.Str)
                Exact = FALSE
                DO WHILE RIGHT$(CurrKey$, 1) = "/"
                        Exact = TRUE
                        CurrKey$ = LEFT$(CurrKey$, LEN(CurrKey$) - 1)
                LOOP

                '-- compare the single keyword [CurrKey$/SingKeyTemp.Str]
                '   against the search word [Term$]

                Match = (Term$ = LEFT$(CurrKey$, LEN(Term$)))

                IF NOT Exact AND NOT Match THEN
                        Match = (CurrKey$ = LEFT$(Term$, LEN(CurrKey$)))
                END IF
                IF Match THEN GOSUB SaveSingStr

        NEXT' key in range
                IF NOT Match THEN

                        SELECT CASE SearchStep
                         CASE 1: Term$ = UCASE$(LEFT$(Term$, 1)) + LCASE$(RIGHT$(Term$, LEN(Term$) - 1))
                                 SearchStep = SearchStep + 1
                         CASE 2:  Term$ = UCASE$(Term$)
                                 SearchStep = SearchStep + 1
                         CASE ELSE: GOTO AddNextKey
                         END SELECT
                         GOTO AddSearchAgain
                 ELSE GOSUB SaveSingStr
                        IF Term$ = CurrKey$ THEN ExactFlag = TRUE
                 END IF
AddNextKey:
END IF' the range was valid

'  FOR j = 1 TO NumSingKey
'     CALL FGetRT(SingKeyFILE, SingKeyTemp, CLNG(j), LEN(SingKeyTemp))
'     CurrKey$ = RTRIM$(SingKeyTemp.Str)
'     DO WHILE RIGHT$(CurrKey$, 1) = "/"
'        CurrKey$ = LEFT$(CurrKey$, LEN(CurrKey$) - 1)
'     LOOP
'     IF CurrKey$ = LEFT$(Term$, LEN(CurrKey$)) THEN
'        GOSUB SaveSingStr
'     END IF
'  NEXT

' Done searching for single keys, go into FindCombKey routine now
'Term$ = Term$ + " "

EXIT SUB

' =-=-=-=
```

```
        IF RIGHT$(Term$, 1) = " " GOTO CombKeyMatched

        ' continue matching the rest of the words in the combined key
        ' exiting out as soon as there's a non-match

        FOR k = 1 TO Words - 1' number of words left in combined key
                ' extract the next word from the current combined keyword (j)
                CALL Extract(CombKeyTemp.Str, " ", k + 1, Strt, Slen)
                CurrKey$ = MID$(CombKeyTemp.Str, Strt, Slen)
                IF RIGHT$(CurrKey$, 1) = "/" THEN ' remove / at end of word
                        Exact = TRUE
                        CurrKey$ = LEFT$(CurrKey$, Slen - 1)
                        Slen = Slen - 1' account for / at end
                ELSE
                        Exact = FALSE
                END IF

                ' get next word to compare
                CALL Extract(Term$, " ", k + 1, Strt, DLen)
                DocWord$ = MID$(Term$, Strt, DLen)

                IF CurrKey$ = "@" THEN ' special processing for @ wildcard
                        IF INSTR(AtList$, "/" + DocWord$ + "/") THEN
                                Match = TRUE' the word was in the @ list, so continue
                        ELSE
                                Match = FALSE
                        END IF
                ELSE
                        Match = (DocWord$ = LEFT$(CurrKey$, LEN(DocWord$)))
                        IF NOT Match THEN
                                IF Exact THEN ' check for *exact* match
                                        Match = (CurrKey$ = DocWord$)
                                ELSE ' wildcard match, only compare # of chars in CurrKey$
                                        Match = (CurrKey$ = LEFT$(DocWord$, Slen))
                                END IF
                        END IF
                END IF

                IF NOT Match THEN EXIT FOR

        NEXT' word in current combined keyword

CombKeyMatched:

        IF Match THEN GOSUB SaveCombStr

NextCombKey:

    NEXT
END IF' Table range was valid

'Term$ = RTRIM$(Term$)
'FOR j = 1 TO NumCombKey
'   CALL FGetRT(CombKeyFILE, CombKeyTemp, CLNG(j), LEN(CombKeyTemp))
'   CurrKey$ = LEFT$(CombKeyTemp.Str, INSTR(CombKeyTemp.Str, " ") - 1)
'   DO WHILE RIGHT$(CurrKey$, 1) = "/"
'      CurrKey$ = LEFT$(CurrKey$, LEN(CurrKey$) - 1)
'   LOOP
'   IF CurrKey$ = LEFT$(Term$, LEN(CurrKey$)) THEN
'      GOSUB SaveCombStr
'   END IF
'NEXT

EXIT SUB


'-------------------------------------------------------------------------
' check if this code was already added by a previous match to a synonym

SaveCombStr:

    '-- don't need the dict lookup, the code is in the .STR structure already
    '** 4/26/91 4:50p TMY
    'DictTemp.Str = CombKeyTemp.Str

    TempCode = CombKeyTemp.Code
    GOTO CheckStr


SaveSingStr:

    '-- don't need the dict lookup, the code is in the .STR structure already
    '** 4/26/91 4:50p TMY
    'DictTemp.Str = SingKeyTemp.Str

    TempCode = SingKeyTemp.Code

CheckStr:

    '-- don't need the dict lookup, the code is in the .STR structure already
    '** 4/26/91 4:50p TMY
    'TempCode = DictSrch%(DictTemp)

    IF KYIndx(TempCode).Num > 0 THEN ' it appeared in this database
            Found = FALSE
            FOR i = 1 TO NumM
                IF m(i).Code = TempCode THEN Found = TRUE: EXIT FOR
            NEXT
```

```
            IF NOT Found THEN 'then add the single keyword to the list of matches
                  '— check to see if it's already in the expression
                  IF KeyInstrZ(Expr.SubExpr(1).Phrase, MKI$(TempCode)) = 0 THEN
                        NumM = NumM + 1
                        REDIM PRESERVE m(1 TO NumM) AS CollectType
                        m(NumM).Code = TempCode
                  ELSE
                        Repeate = TRUE
                  END IF
            END IF
      END IF

      RETURN

END SUB
```

What is claimed is:

1. A method of indexing and retrieving documents, said method using a digital computer system having a central processing unit, a memory, a display screen, a keyboard, and a large capacity file system, said method comprising the steps of:

(a) storing in said memory a vocabulary of terms, each term consisting of one or more words, and for each term an associated term-code;

(b) storing on said file system a collection of documents each with an associated unique document-number;

(c) creating index files which contain for each said term-code in (a)

(i) the set of document-numbers in (b) such that the corresponding documents contain the corresponding term; and

(ii) for each said document-identifying-number in (i) the frequency-in-document of the corresponding term which is the number of times that said term appears in the corresponding document;

(d) creating a weight-in-document file which contains for each document-number in (c)(i) the weight-in-document of the corresponding term which is calculated using the frequency-in-document in (c) (ii), the number of document-numbers in (c) (i), and the total number of terms in (a) which are in the corresponding document (counted multiple times);

(e) creating a frequent-companion file which contains for each occurring term-code in (a) a ranked set of pairs of numbers where each pair consists of a first element term-code and a second element companion-percentage, where the companion-percentage is calculated by summing the weight-in-document values of said first element term-code over documents that contain both the term corresponding to said first element term-code and the term corresponding to said occurring term-code and then dividing by the sum over all documents of the weight-in-document of said occurring term-code;

(f) creating a relative file which contains for each occurring term-code in (a) a ranked set of pairs of numbers where each pair consists of a first element relative term-code and a second element relative-percentage, where the relative-percentage is calculated by taking a weighted average of the companion-percentage of said first element term-code calculated in step (e) and the companion-percentage of said occurring term-code that was calculated in step (e) when said first element term-code was the occurring term-code and said occurring term-code was the first element term-code;

(g) creating a polysemantic file which contains for each occurring term-code in (a), a polysemantic weight which is calculated using the number of sets of pairs in the relative file created in step (f) that said occurring term-code appears in, the number of documents-numbers for which the weight-in-document of said occurring term-code calculated in step (d) is greater than some threshold value, and the averages for several values of N of the first N relative-percentages of said occurring term-code calculated and ranked in step (f);

(h) accepting a query consisting of a sequence of words entered by a user using said keyboard and creating a parsed-query table of term-codes which consist of the term-codes in said vocabulary that are associated with the terms that are contained in said query;

(i) creating a temporary swap table of pairs of first element term-codes and corresponding second element summed-relative-percentages consisting of those relative term-codes created in step (f) where said corresponding second element summed-relative-percentages are the sum, over all said occurring term-codes that are in said parsed-query table, of the relative percentages of said first element term-codes;

(j) creating a modified swap table by modifying said second element summed-relative-percentages created in step (i) by multiplying them by a function of the polysemantic weight of the corresponding first element term-codes;

(k) sorting said modified swap table by said modified summed-relative-percentages in descending order;

(l) displaying on said display the terms corresponding to the term-codes of said modified swap table;

(m) accepting user keypresses or other actions which identify one or more of the terms displayed in step (l) and adding the corresponding term-codes to the parsed-query-table;

(n) repeating steps (i) through (m) as many times as the user indicates by his input;

(o) accepting an input from the user indicating a command to retrieve documents;

(p) creating a temporary rank table of pairs of first element document-numbers and corresponding second element summed-document-weight×poly values which pairs comprise those document-numbers for which any of the term-codes that are in said parsed-query table have weight-in-document above a threshold value, and summed-document-weight×poly values which are the sums, over all term-codes in said parsed-query table, of a function of me polysemantic weight of the term-code and the weight-in-document of the term-code;

(r) creating a sorted rank table by sorting said temporary rank table by the value of the second elements of the pairs in descending order;

(s) displaying on the display screen some portion of the document corresponding to the first document number in the sorted rank table and some indication of the corresponding summed-document-weight×poly value;

(t) displaying other documents corresponding to other document-numbers in the sorted rank table in response to inputs from the user.

2. A method as in claim 1 wherein additional steps (j)(l) and (p)(l) are carried out after steps (j) and (p) respectively to implement the soft boolean connector algorithm which consists of the following steps:

(A) creating a table of relative penalties for each pair of said term-codes in said parsed-query table where said relative penalty is a function of the relative percentage corresponding to the two term-codes of said pair, the number of documents that each of the term-codes of the pair are contained in with a document-weight above a threshold, and the average over all terms of the number of documents that the term is contained in with a document-weight above said threshold;

(B) modifying said relative penalties by taking the minimum of the relative penalty and some maximum value which depends on the number of terms in the parsed-query table;

(C) summing said modified relative penalties to produce a sum of relative penalties;

(D) modifying said sum of relative penalties by taking the minimum of said sum and some maximum sum value which depends on the number of terms in the parsed-query table to produce a modified sum of penalties;

(E) summing some function of the polysemantic weights of the term-codes in the parsed-query table that are either relatives of a potential SWAPS term (jl) or are contained in a document (pl) to produce a number of hits value;

(F) Calculating some function of the number of hits value and the modified sum of penalties value to produce a power value;

(G) Raising a number approximately equal to 2 to the power value to produce an adjust value;

(H) Multiplying either the modified summed relative percentages calculated in step j) or the summed document weight×poly values calculated in step (p) by the adjust value.

3. A method as in claim 1 where the formula for calculating the weight-in-document in step (d) is:

$$\text{Weight(Word)} = \frac{\text{Log}_2(\text{FreqInDoc} + 1) \times \text{Log}_2\left(\frac{\text{TotDocs} \times 1.5}{\text{DocsWithWord} + 3 + \frac{\text{TotDocs}}{1000}}\right)}{\text{Log}_2\left(2 + \frac{\text{TotalKeywordsInDoc}}{10}\right)}$$

4. A method as in claim 1 where the formula for calculating the polysemantic weight in step (g) is:

$$\text{PolyValue} = \sqrt{\left(\text{Avg}_3 \times \frac{\text{Avg}_3}{\text{Avg}_{20}} + \text{Avg}_6 \times \frac{\text{Avg}_6}{\text{Avg}_{63}}\right)} \times \sqrt{\frac{\text{TotRelVal}}{\text{DocFreq}^{1.2}}}$$

5. A method as in claim 1 where the function in step (j) is the identity function.

6. A method as in claim 1 where the function in step (p) is the identity function.

* * * * *

US005488725A

United States Patent [19]

Turtle et al.

[11] Patent Number: 5,488,725

[45] Date of Patent: Jan. 30, 1996

[54] **SYSTEM OF DOCUMENT REPRESENTATION RETRIEVAL BY SUCCESSIVE ITERATED PROBABILITY SAMPLING**

[75] Inventors: **Howard R. Turtle**, Rosemount; **Gerald J. Morton**, South St. Paul; **F. Kinley Larntz**, Shoreview, all of Minn.

[73] Assignee: **West Publishing Company**, Eagan, Minn.

[21] Appl. No.: **39,757**

[22] Filed: **Mar. 30, 1993**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 773,101, Oct. 8, 1991, Pat. No. 5,265,065.

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **395/600**; 364/419.19; 364/DIG. 1; 364/282.1; 364/282.3

[58] **Field of Search** 395/600; 364/419.19

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,384,329	5/1983	Rosenbaum et al.	364/300
4,422,158	12/1983	Galie	395/400
4,554,631	11/1985	Reddington	364/283.2
4,843,389	6/1989	Lisle et al.	341/106
4,870,568	9/1989	Kahle et al.	395/600
5,109,509	4/1992	Katayama et al.	395/600
5,159,667	10/1992	Borrey et al.	395/148
5,220,625	6/1993	Hatakeyama et al.	382/54
5,263,159	11/1993	Mitsui	395/600
5,265,065	11/1993	Turtle	395/600
5,278,980	1/1994	Pedersen et al.	395/600
5,297,042	3/1994	Morita	364/419.19
5,301,109	4/1994	Landauer et al.	364/419.19
5,321,833	6/1994	Chang et al.	395/600
5,325,298	6/1994	Gallant	364/419.19
5,335,345	8/1994	Frieder et al.	395/600
5,418,948	5/1995	Turtle	395/600

OTHER PUBLICATIONS

M. E. Smith, "Aspects of the P–Norm Model of Information Retrieval: Syntactic Query Generation, Efficiency, and Theoretical Properties" Ph. D Thesis, Department of Computer Science, Cornell University, Ithaca, NY., TR 90–1128 (May, 1990), pp. 116–120.

Buckley et al., "Optimization of Inverted Vector Searches", *Proceedings of the Association for Computing Machinery (SIGIR 85)*, 1985, pp. 97–110.

Turtle et al., "Uncertainty in Information Retrieval Systems", *Provisional Proceedings on Uncertainty Management in Information Systems*, sponsored by the National Science Foundation and ESPIRIT, Majorca, Spain, Sep. 23, 1992, pp. 111–137.

Croft et al, "A Retrieval Model Incorporating Hypertext Links", *Hypertex '89 Proceedings*, Association for Computer Machinery, pp. 213–224 (Nov 1989).

Turtle et al, "Inference Networks for Document Retrieval",

(List continued on next page.)

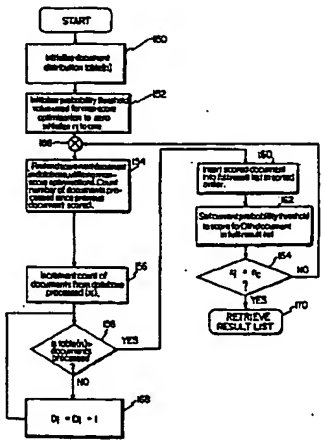
Primary Examiner—Thomas G. Black
Assistant Examiner—Wayne Amsbury
Attorney, Agent, or Firm—Kinney & Lange

[57] **ABSTRACT**

An information retrieval system based on probabilities that documents meet information needs. The frequency of occurrence of a representation in a collection of documents is estimated by identifying the frequency of occurrence of the representation in a sample of documents and calculating the difference between the maximum and minimum probable frequencies of occurrence of the representation in the collection. If the difference does not exceed a limit, a midpoint of the maximum and minimum probable frequencies is the estimated frequency of occurrence of the representation.

Document distribution probabilities are optimized and probability thresholds are established for the identification of documents. An initial probability threshold is established and is adjusted as the probabilities are scored for documents in samples. The document result list is iteratively adjusted through the samples.

46 Claims, 13 Drawing Sheets



OTHER PUBLICATIONS

COINS Technical Report 90–07, *University of Massachusetts* (Mar. 1990).

Turtle et al, "Inference Network for Document Retrieval", *SIGIR 90,* Association for Computing Machinery, pp. 1–24 (Sep. 1990).

Turtle, "Inference Network for Document Retrieval", Ph.D. Dissertation, *COINS Technical Report 90–92,* University of Massachusetts (Oct. 1990).

Turtle et al, "Efficient Probabilistic Inference for Text Retrieval", *RIAO '91 Conference Proceedings,* Recherche d'Informaion Assistée par Ordinateur, Universitat Autónoma de Barcelona, Spain, pp. 644–661 (Apr. 1991).

Turtle et al., "Evaluation of an Inference Network–Based Retrieval Model", *Transactions on Information Systems,* Association for Computer Machinery, vol. 9, No. 3. pp. 187–223 (Jul. 1991).

Croft et al., "Interactive Retrieval of Complex Documents", *Information Processing and Management,* vol. 26, No. 5. pp. 593–613 (1990).

Haynes, "Designing a System for the Specialized User: A Case Study", *Proceedings—1985 National Online Meeting,* Learning Information Inc., pp. 205–213, Apr. 30, 1985.

Croft et al., "The Use of Phrases and Structured Queries in Information Retrieval", *SIGIR 91,* Association for Computing Machinery, pp. 1–23 (Oct. 13, 1991).

Turtle et al., "A Comparison of Text Retrieval Models", *The Computer Journal,* vol. 35, No. 3, pp. 279–290 (1992).

Croft et al., "Retrieval of Complex Objects", *Proceedings of the International Conference on Extending Database Technology,* pp. 217–229 (Mar. 1992).

Croft et al., "Text Retrieval and Inference", *Text Based Intelligent Systems,* pp. 127–155 (1992).

Croft et al., "A Loosely–Coupled Integration of a Text Retrieval System and an Object–Oriented Database System", *SIGIR 92,* Association for Computing Machinery, pp. 223–232 (1992).

Porter, "An Algorithm for Suffix Skipping", *Program,* vol. 15, pp. 130–137 (1980).

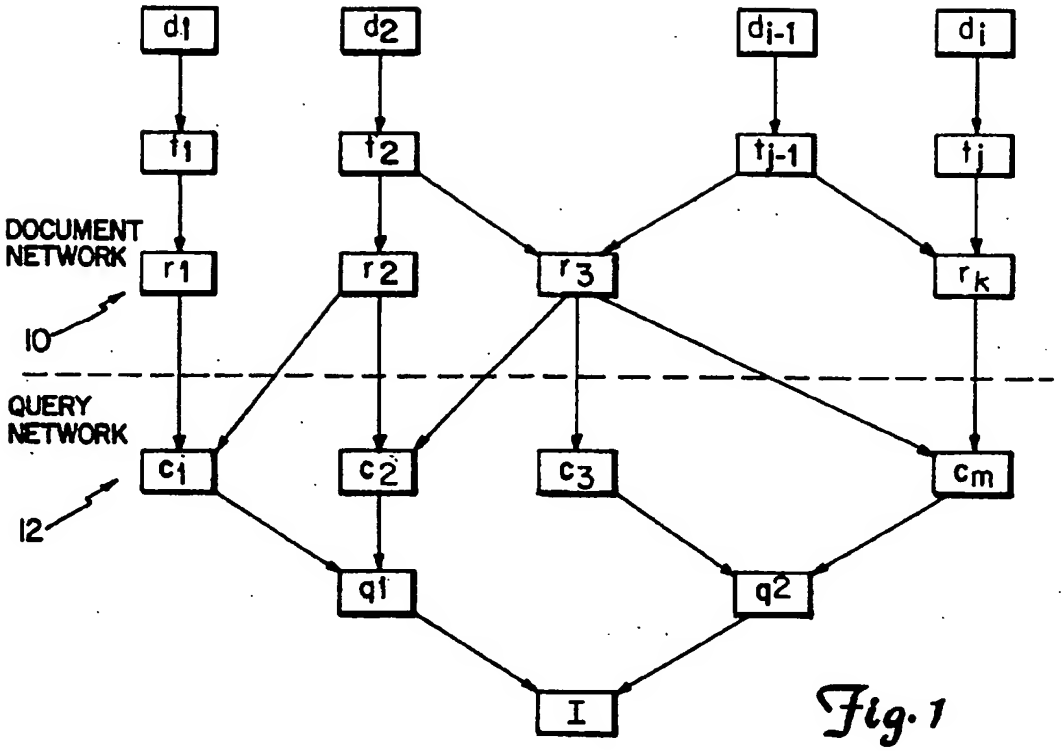

d1
d2
di-1
di
t1
t2
tj-1
tj
DOCUMENT NETWORK
r1
r2
r3
rk
I0
QUERY NETWORK
c1
c2
c3
cm
I2
q1
q2
I

Fig. 1

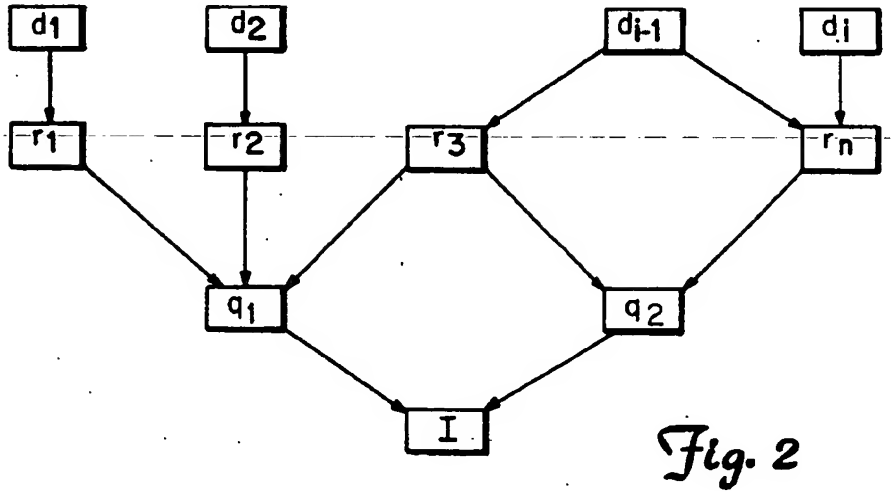

d1
d2
di-1
di
r1
r2
r3
rn
q1
q2
I

Fig. 2

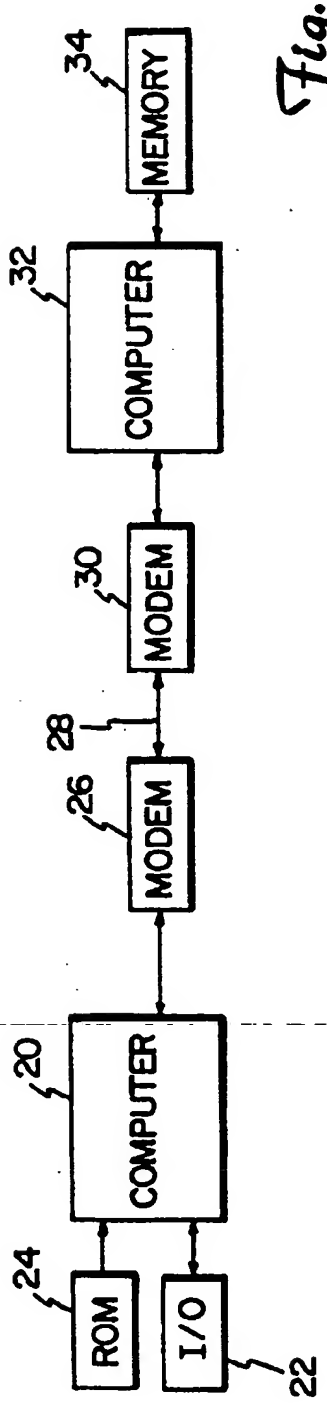

34
MEMORY
32
COMPUTER
30
MODEM
28
26
MODEM
20
COMPUTER
24
ROM
I/O
22

Fig. 3

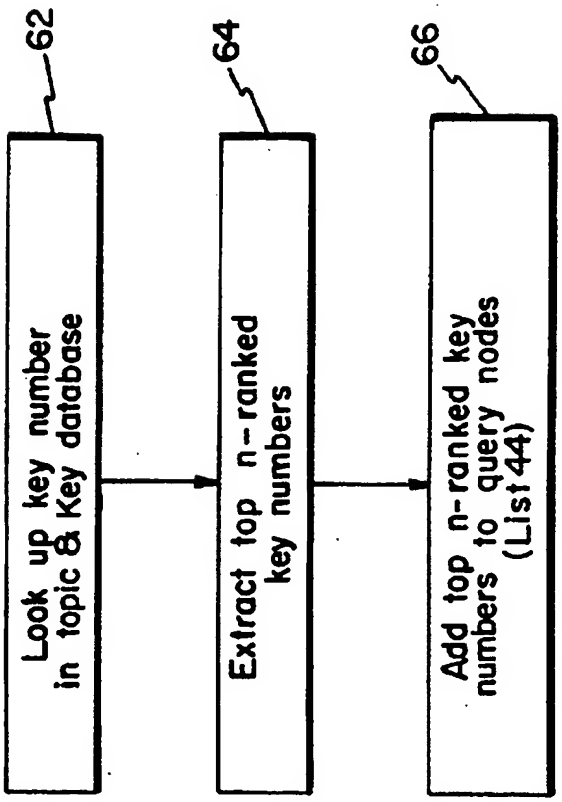

62
Look up key number in topic & Key database
64
Extract top n-ranked key numbers
66
Add top n-ranked key numbers to query nodes (List 44)

Fig. 5

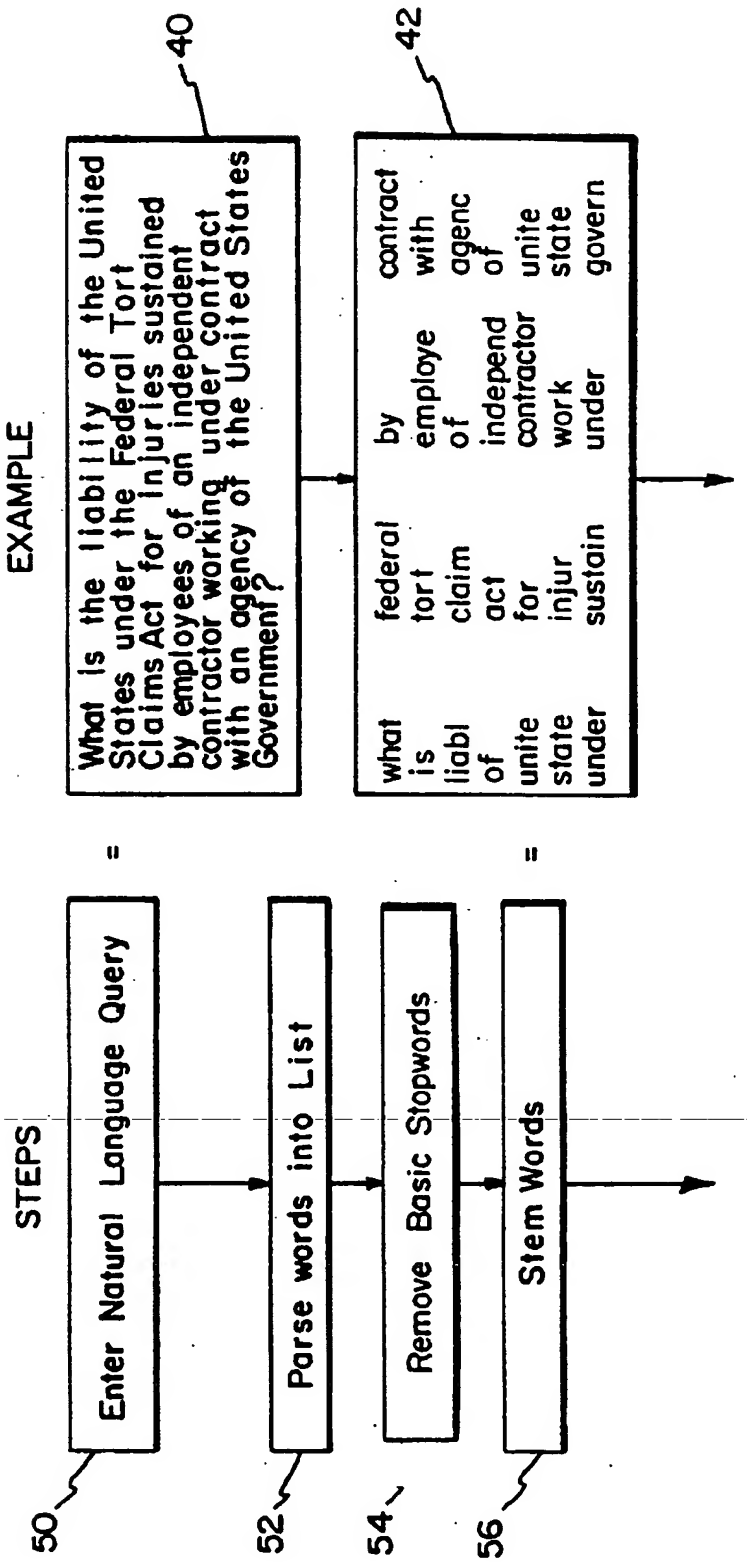
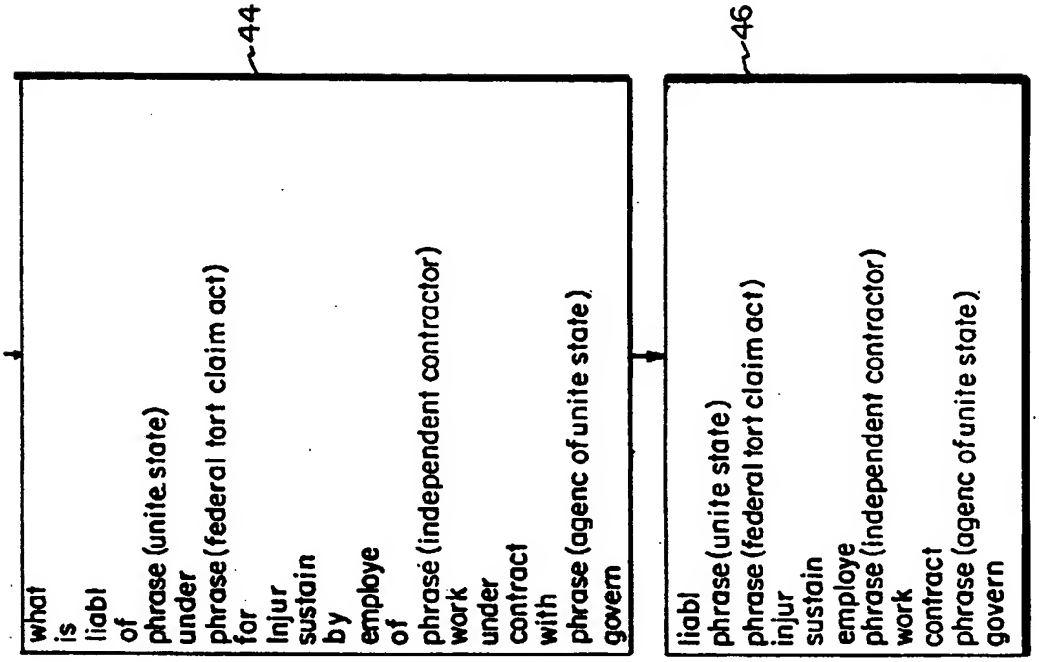

EXAMPLE
40
42
What is the liability of the United States under the Federal Tort Claims Act for injuries sustained by employees of an independent contractor working under contract with an agency of the United States Government?
what is liabl of unite state under
federal tort claim act for injur sustain
by employe of independ contractor work under
contract with agenc of unite state govern
=
=
STEPS
Enter Natural Language Query
Parse words into List
Remove Basic Stopwords
Stem Words
50
52
54
56

Fig. 4A


44
what
is
liabl
of
phrase (unite state)
under
phrase (federal tort claim act)
for
injur
sustain
by
employe
of
phrase (independent contractor)
work
under
contract
with
phrase (agenc of unite state)
govern
46
liabl
phrase (unite state)
phrase (federal tort claim act)
injur
sustain
employe
phrase (independent contractor)
work
contract
phrase (agenc of unite state)
govern

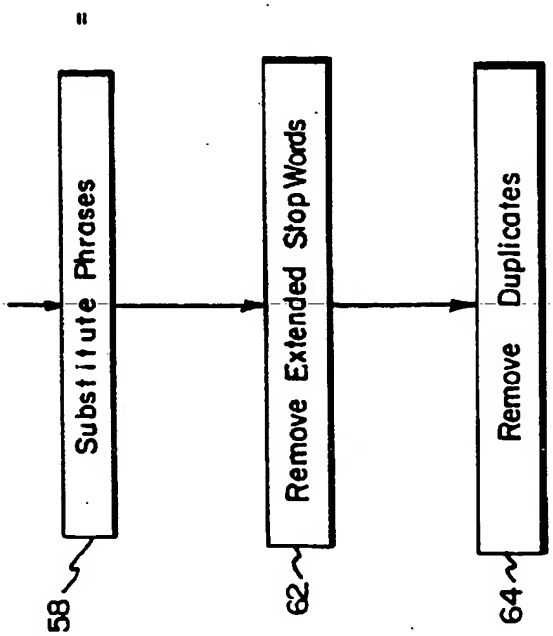

=
Substitute Phrases
Remove Extended Stop Words
Remove Duplicates
58
62
64

Fig. 4B

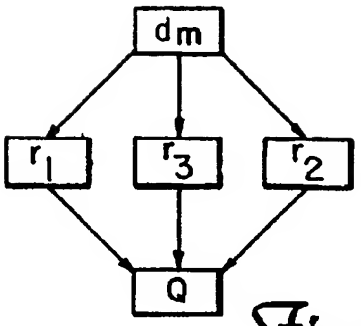


Fig 6A

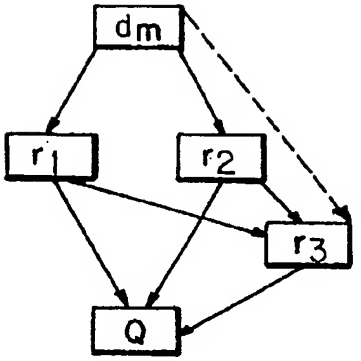


Fig. 6B

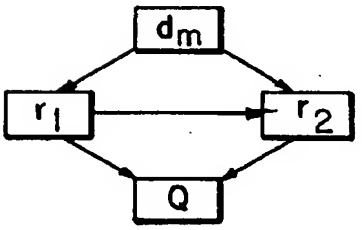


Fig. 6C

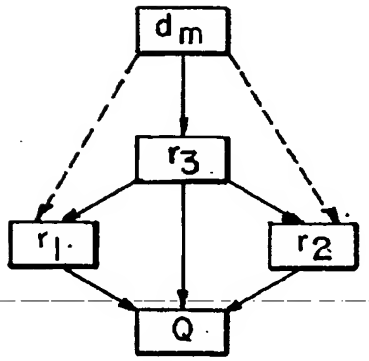


Fig. 6D

Fig. 7A

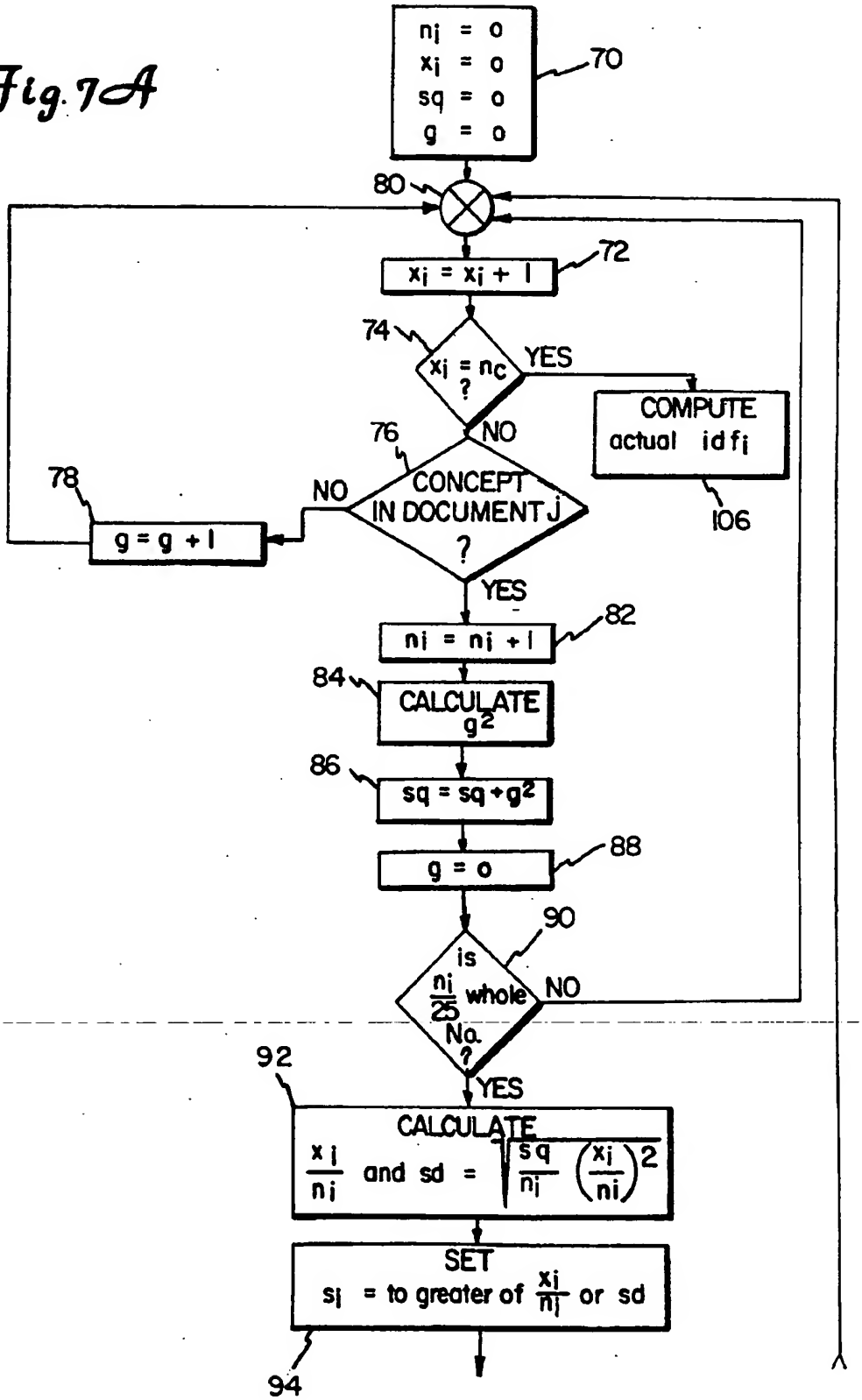


Fig. 7B

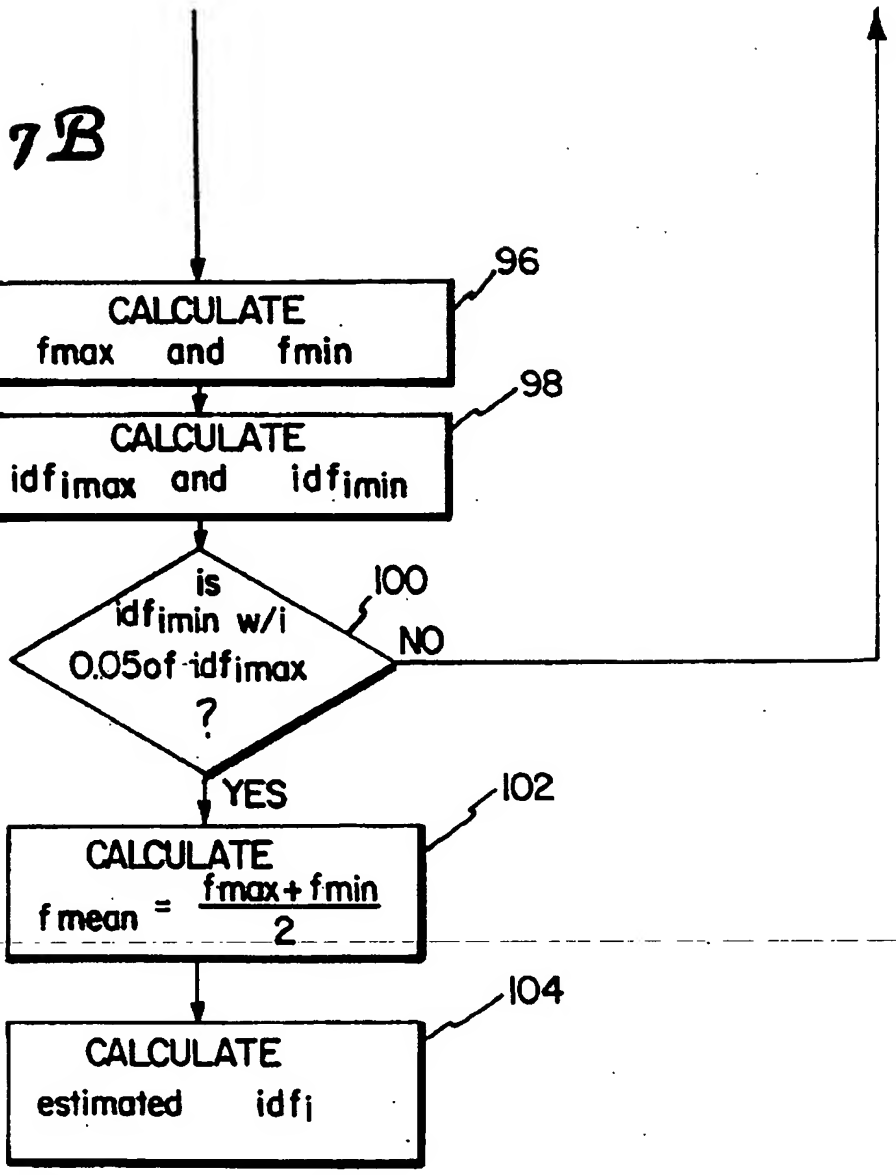

96
CALCULATE
fmax and fmin
98
CALCULATE
idfimax and idfimin
100
is
idfimin w/i
0.05of idfimax
?
NO
YES
102
CALCULATE
fmean = fmax+fmin / 2
104
CALCULATE
estimated idfi

Fig. 8

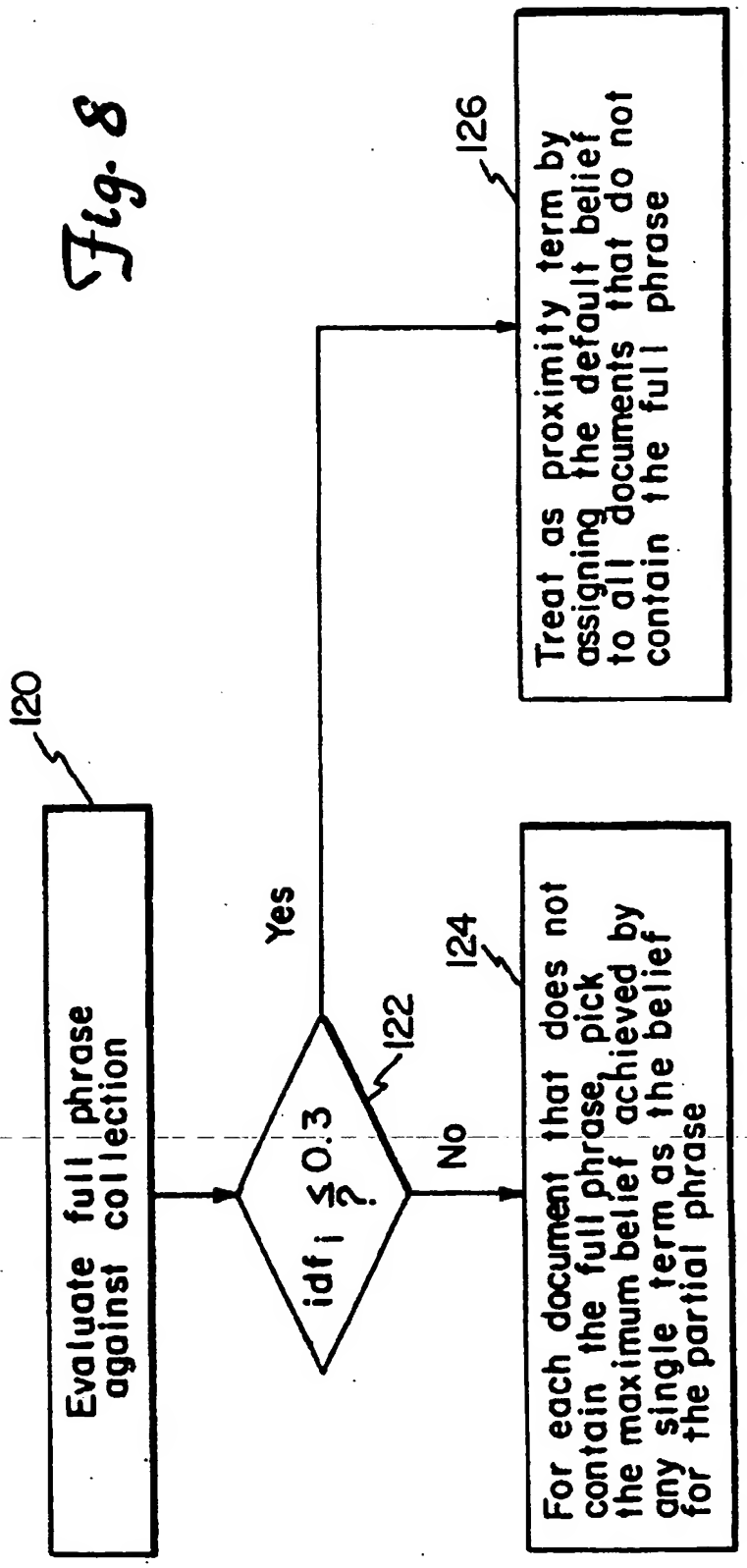
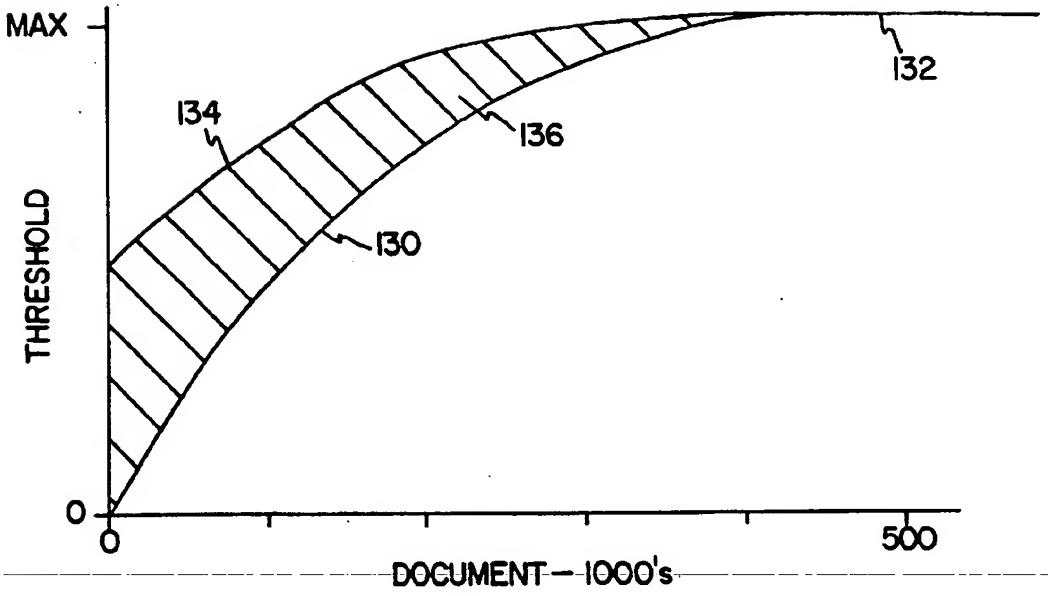

120
Evaluate full phrase against collection
122
idf_i ≤ 0.3 ?
Yes
No
126
Treat as proximity term by assigning the default belief to all documents that do not contain the full phrase
124
For each document that does not contain the full phrase, pick the maximum belief achieved by any single term as the belief for the partial phrase

Fig. 9



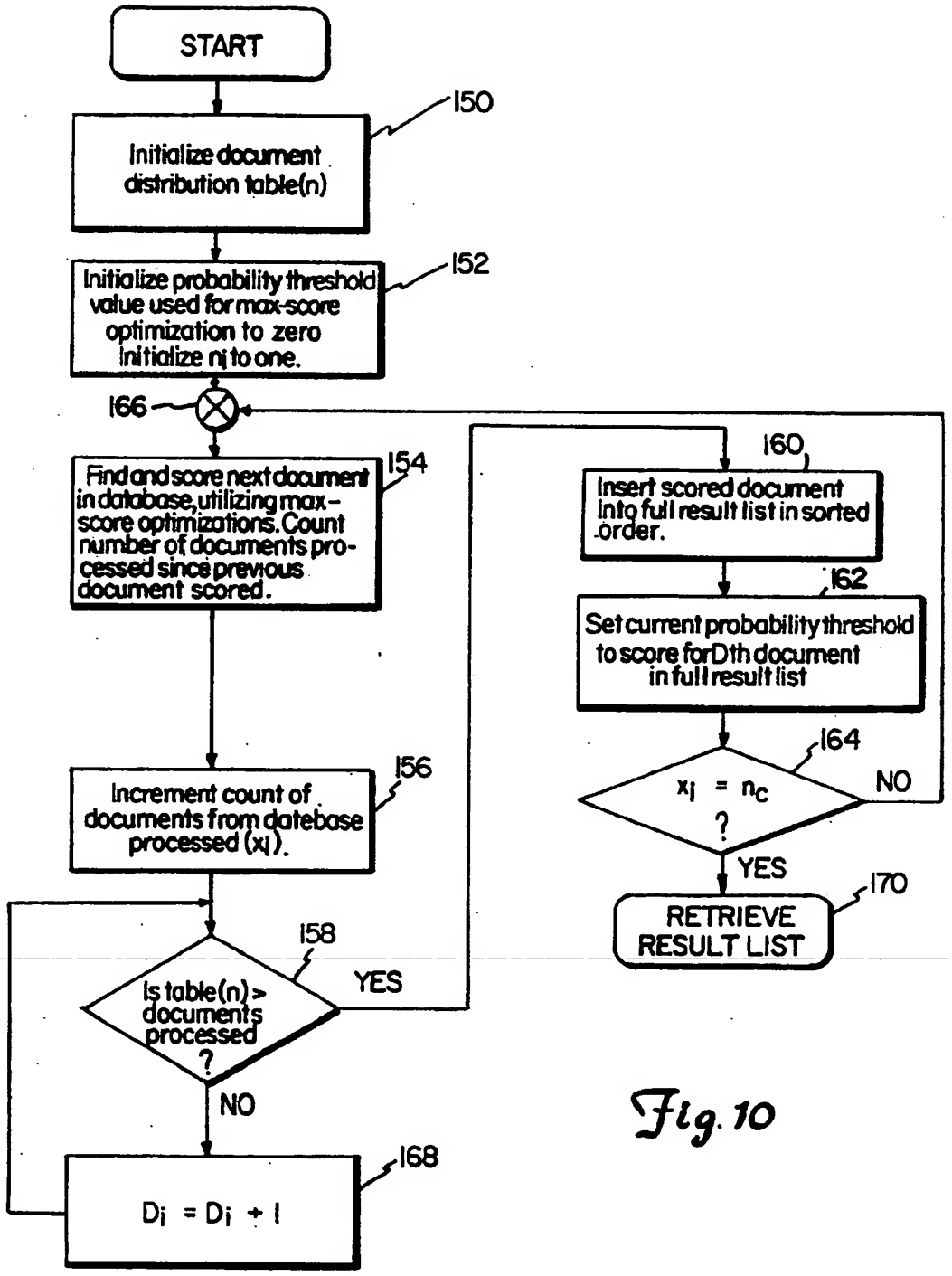

START
150
Initialize document distribution table(n)
152
Initialize probability threshold value used for max-score optimization to zero Initialize n_i to one.
166
154
Find and score next document in database, utilizing max-score optimizations. Count number of documents processed since previous document scored.
156
Increment count of documents from database processed (x_i).
158
Is table(n) > documents processed ?
YES
NO
168
$D_i = D_i + 1$
160
Insert scored document into full result list in sorted order.
162
Set current probability threshold to score for Dth document in full result list
164
$x_i = n_c$?
NO
YES
170
RETRIEVE RESULT LIST

Fig. 10

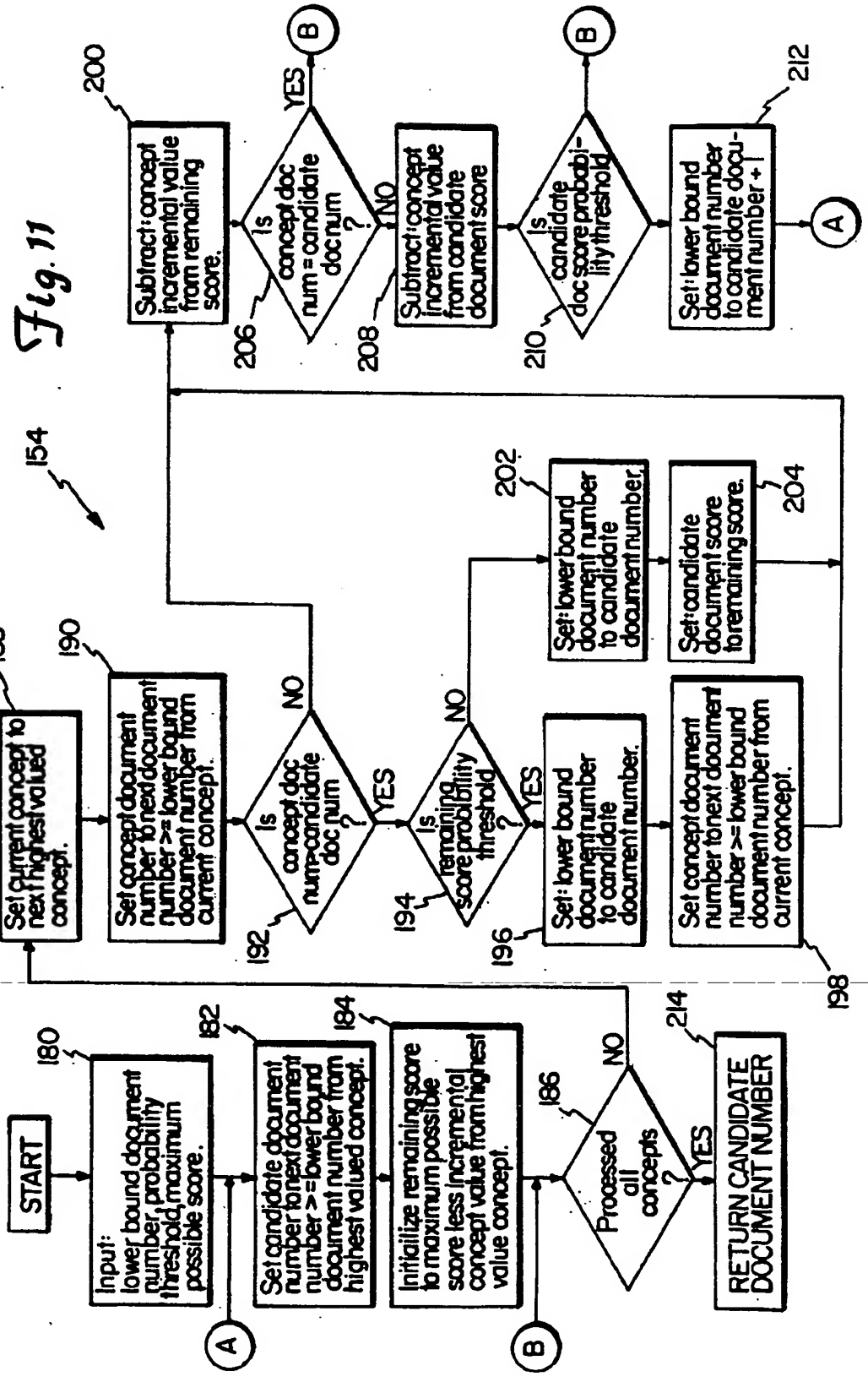

START
Input: lower bound document number, probability threshold, maximum possible score.
Set candidate document number to next document number >= lower bound document number from highest valued concept.
Initialize remaining score to maximum possible score less incremental concept value from highest value concept.
Processed all concepts ?
NO
YES
RETURN CANDIDATE DOCUMENT NUMBER
Set current concept to next highest valued concept.
Set concept document number to next document number >= lower bound document number from current concept.
Is concept doc num = candidate doc num ?
Is remaining score probability threshold ?
Set: lower bound document number to candidate document number.
Set concept document number to next document number >= lower bound document number from current concept.
Set: lower bound document number to candidate document number.
Set: candidate document score to remaining score.
Subtract: concept incremental value from remaining score.
Is concept doc num = candidate doc num ?
Subtract: concept incremental value from candidate document score
Is candidate doc score probability threshold
Set: lower bound document number to candidate document number + 1
A
B
154
180
182
184
186
188
190
192
194
196
198
200
202
204
206
208
210
212
214

Fig. 11

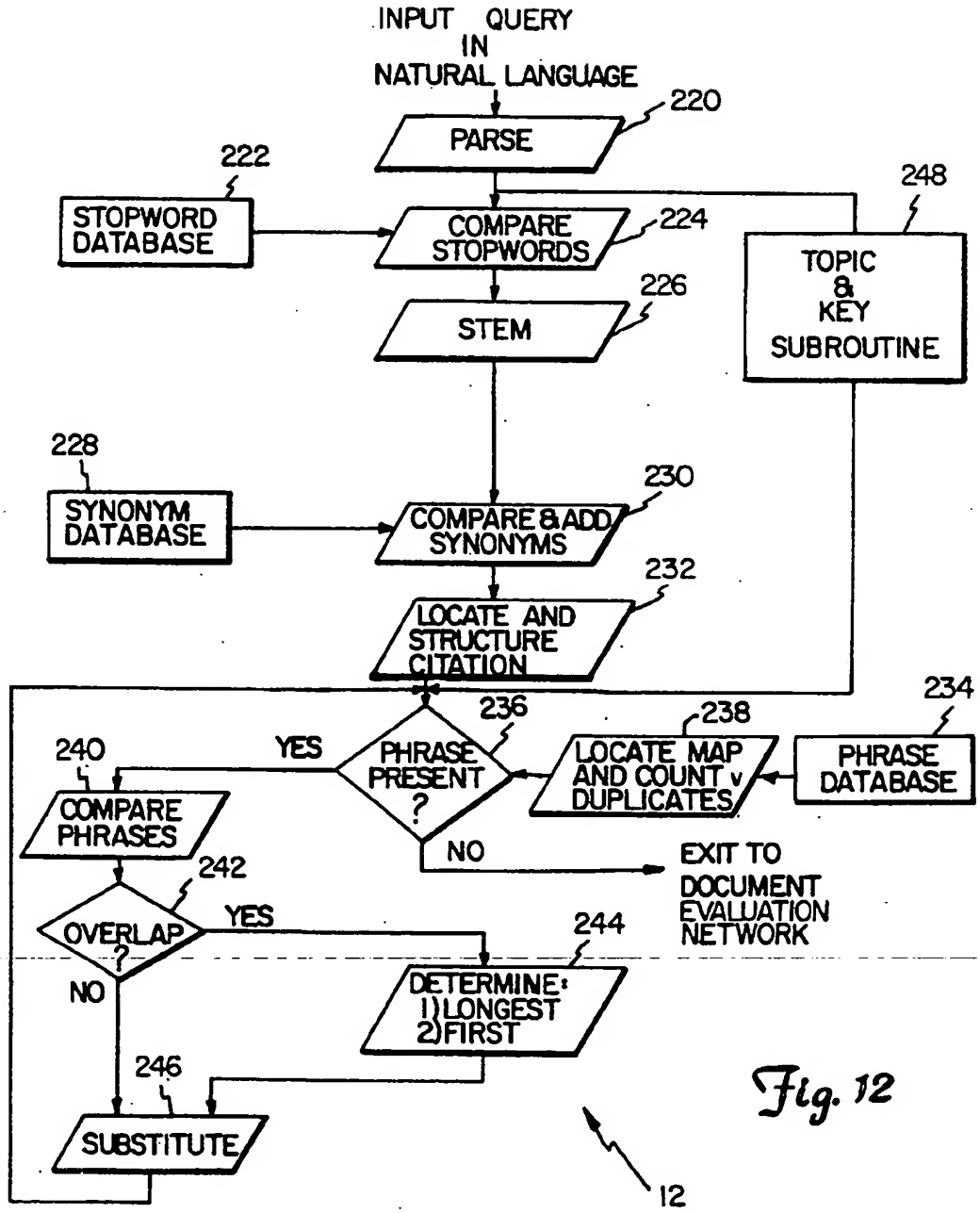

INPUT QUERY IN NATURAL LANGUAGE
220
PARSE
222
STOPWORD DATABASE
COMPARE STOPWORDS
224
248
TOPIC & KEY SUBROUTINE
226
STEM
228
SYNONYM DATABASE
230
COMPARE & ADD SYNONYMS
232
LOCATE AND STRUCTURE CITATION
234
236
238
240
YES
PHRASE PRESENT ?
LOCATE MAP AND COUNT v DUPLICATES
PHRASE DATABASE
COMPARE PHRASES
NO
EXIT TO DOCUMENT EVALUATION NETWORK
242
OVERLAP ?
YES
244
NO
DETERMINE: 1) LONGEST 2) FIRST
246
SUBSTITUTE
12

Fig. 12

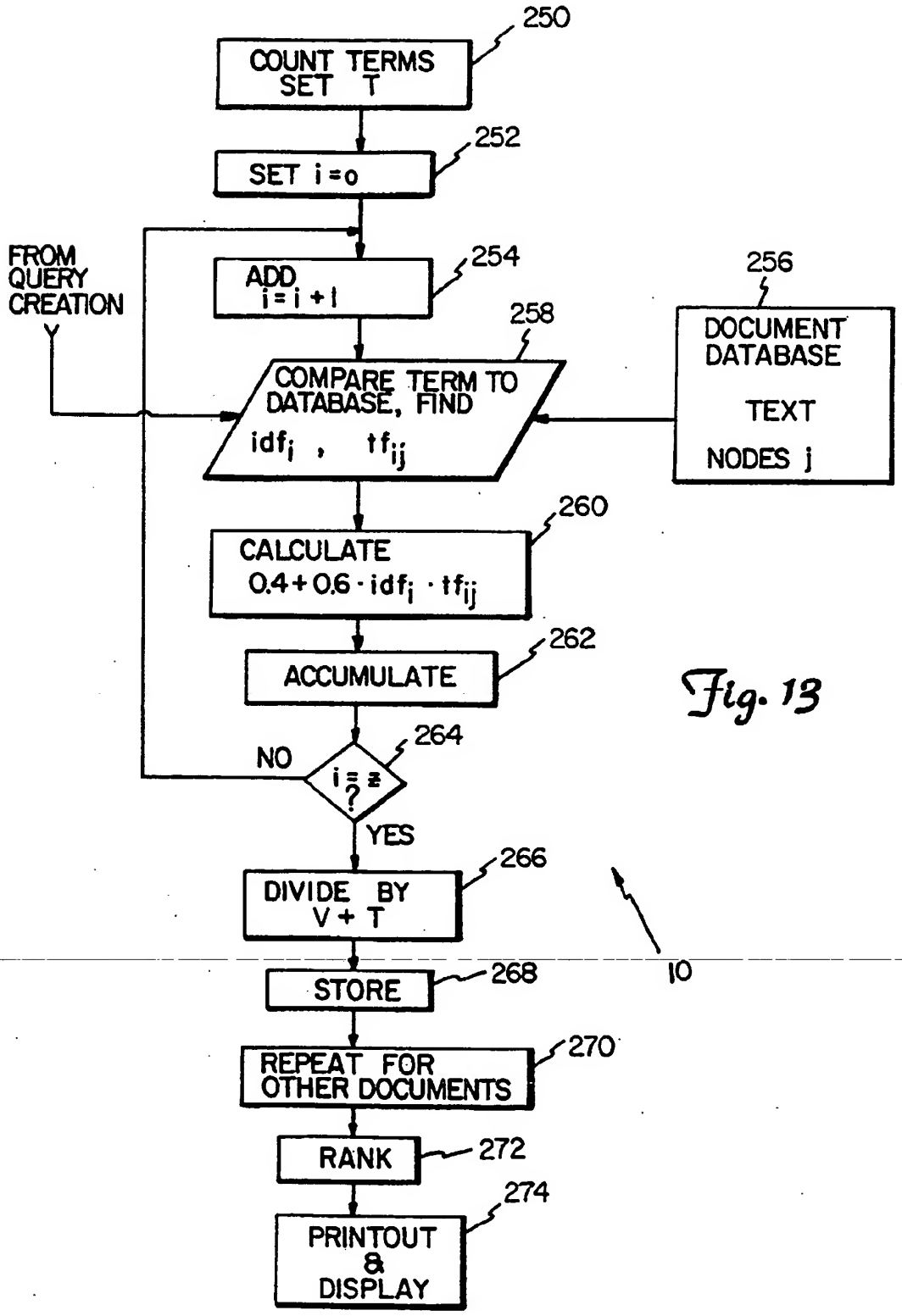

250
COUNT TERMS
SET T
252
SET i=o
FROM
QUERY
CREATION
Y
254
ADD
i=i+1
256
DOCUMENT
DATABASE
TEXT
NODES j
258
COMPARE TERM TO
DATABASE, FIND
idf_i , tf_{ij}
260
CALCULATE
$0.4+0.6 \cdot idf_i \cdot tf_{ij}$
262
ACCUMULATE
264
NO
i = z
?
YES
266
DIVIDE BY
V + T
10
268
STORE
270
REPEAT FOR
OTHER DOCUMENTS
272
RANK
274
PRINTOUT
&
DISPLAY

Fig. 13

SYSTEM OF DOCUMENT REPRESENTATION RETRIEVAL BY SUCCESSIVE ITERATED PROBABILITY SAMPLING

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of application Ser. No. 07/773,101 filed Oct. 8, 1991, U.S. Pat. No. 5,265,065.

BACKGROUND OF THE INVENTION

This invention relates to information retrieval, and particularly to document retrieval from a computer database using probability techniques. More particularly, the invention concerns a method and apparatus for establishing probability thresholds in probabilistic information retrieval systems and for estimating representation frequencies in document databases for representations having no pre-computed frequency.

There are, in theory, two categories of information retrieval systems: algebraic systems and probabilistic systems. Algebraic systems logically match terms and their positions in a stored information (such as a document) to terms in a query; Boolean systems are examples of algebraic systems. Probabilistic systems match representations (concepts) in a stored information to concepts in a query to retrieve information based on probabilities rather than algebraic or Boolean logic.

Presently, document retrieval is most commonly performed through use of Boolean search queries to search the texts of documents in the database. These retrieval systems specify strategies for evaluating documents with respect to a given query by logically comparing search queries to document texts. One of the problems associated with text searching is that for a single natural language description of an information need, different Boolean researchers will formulate different Boolean queries to represent that need. Because the queries are different, different documents will be retrieved for each search.

Another difficulty with Boolean systems is that all documents meeting the query are retrieved, regardless of number. If an unmanageable number of documents are retrieved, the searcher must reformulate the search query to more narrowly define the information need, thereby narrowing the retrieved documents to a more manageable number. However, in narrowing the search, the researcher risks missing relevant documents partially meeting the information need. Moreover, Boolean systems will not retrieve documents only partially meeting the query, which themselves are often important secondary documents to the query.

More recently, probabilistic systems employing hypertext databases have been developed which emphasize flexible organizations of multimedia "nodes" through connections made with user-specified links and interfaces which facilitate browsing in the network. Early networks employed query-based retrieval strategies to form a ranked list of candidate "starting points" for hypertext browsing. Some systems employed feedback during browsing to modify the initial query and to locate additional starting points. Network structures employing hypertext databases have used automatically and manually generated links between documents and the concepts or terms that are used to represent their content. For example, "document clustering" employs links between documents that are automatically generated by comparing similarities of content. Another technique is "citations" wherein documents are linked by comparing similar citations in them. "Term clustering" and "manually-generated thesauri" provide links between terms, but these have not been altogether suitable for document searching on a reliable basis.

Deductive databases have been developed employing facts about the nodes, and current links between the nodes. A simple query in a deductive database, where N is the only free variable in formula W, is of the form {N|W(N)}, which is read as "Retrieve all nodes N such that W(N) can be shown to be true in the current database." However, deductive databases have not been successful in information retrieval. Particularly, uncertainty associated with natural language affects the deductive database, including the facts, the rules, and the query. For example, a specific concept may not be an accurate description of a particular node; some rules may be more certain than others; and some parts of a query may be more important than others. For a more complete description of deductive databases, see Croft et al. "A Retrieval Model for Incorporating Hypertext Links", Hypertext '89 Proceedings, pp 213–224, November 1989 (Association for Computing Machinery), incorporated herein by reference.

A Bayesian network is a probabilistic network which employs nodes to represent the document and the query. If a proposition represented by a parent node directly implies the proposition represented by a child node, an implication line is drawn between the two nodes. If-then rules of Bayesian networks are interpreted as conditional probabilities. Thus, a rule A→B is interpreted as a probability P(B|A), and the line connecting A with B is logically labeled with a matrix that specifies P(B|A) for all possible combinations of values of the two nodes. The set of matrices pointing to a node characterizes the dependence relationship between that node and the nodes representing propositions naming it as a consequence. For a given set of prior probabilities for roots of the network, the compiled network is used to compute the probability or degree of belief associated with the remaining nodes.

An inference network is one which is based on a plausible or non-deductive inference. One such network employs a Bayesian network, described by Turtle et al. in "Inference Networks for Document Retrieval" SIGIR 90, pp. 1–24 Sep. 1990 (Association for Computing Machinery), incorporated herein by reference. The Bayesian inference network described in the Turtle et al. article comprises a document network and a query network. The document network represents the document collection and employs document nodes, text representation nodes and content representation nodes. A document node corresponds to abstract documents rather than their specific representations, whereas a text representation node corresponds to a specific text representation of the document. A set of content representation nodes corresponds to a single representation technique which has been applied to the documents of the database.

The query network of the Bayesian inference network described in the Turtle et al. article employs an information node identifying the information need, and a plurality of concept nodes corresponding to the concepts that express that information need. A plurality of intermediate query nodes may also be employed where multiple queries are used to express the information requirement.

The Bayesian inference network described in the Turtle et al. article has been quite successful for small, general purpose databases. However, it has been difficult to formu-

late the query network to develop nodes which conform to the document network nodes. More particularly, the inference network described in the Turtle et al, article did not use domain-specific knowledge bases to recognize phrases, such as specialized, professional terms, like jargon traditionally associated with specific professions, such as law or medicine.

One important aspect to probabilistic retrieval networks, such as a Bayesian inference network, is the identification of the frequency of occurrence of a representation in each document and in the entire document collection. A representation that occurs frequently in a document is more likely to be a good descriptor of that document's content. A representation that occurs infrequently in the collection is more likely to be a good discriminator than one that occurs in many documents. Consequently, when creating a database for a probabilistic network, care is taken to identify the representations (content concepts) in the documents, as well as their frequencies. However, it is not always possible to identify certain representations (such as phrases, proximities and thesaurus or synonym classes) or their frequency when creating the database. More particularly, phrases are usually comprised of multiple words which themselves are individual concepts or representations. The concept or representation of a phrase might be different from the concepts or representations of the individual words forming the phrase. For example, the phrase "independent contractor" is a different concept than either of the constituent words "independent" and "contractor". Since it is not always possible to identify all possible phrases, or their frequency of occurrence, during creation of the database, the use of phrases as a matching term in probabilistic networks has not been altogether successful. Proximities (such as citations) and thesaurus and synonym classes have likewise not been successful identifiers because of the inability to identify all synonyms, proximities and thesaurus classes during creation of the database or to pre-assign their frequencies.

Techniques have been developed to identify phrases, synonyms, proximities and thesaurus classes as concepts in the query, and to find phrases, synonyms, proximities and thesaurus classes as representations in the documents. However, no satisfactory technique exists for identifying the frequencies of occurrence of representations in the documents and in the collection when the document collection is large and the frequencies of occurrence are not included in the database.

Another difficulty with probabilistic networks is that for large databases, for example databases containing about one-half million documents or more, the processing resources required to evaluate a query have been too great to be commercially feasible. More particularly, probabilistic networks required that all representations for all documents in the collection containing at least one query term must be examined against all of the concepts in the query. Hence, probabilistic networks required extensive computing resources. While such computing resources might be reasonable for small collections of documents, they were not for large databases. There is, accordingly, a need to improve the processing of probabilistic networks to more efficiently employ the processing resources.

For a more general discussion concerning inference networks, reference may be made to *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* by J. Pearl, published by Morgan Kaufmann Publishers, Inc., San Mateo, Calif., 1988, and to *Probabilistic Reasoning in Expert Systems* by R. E. Neapolitan, John Wiley & Sons, New York, N.Y., 1990.

GLOSSARY

As used herein, the following alpha-numeric characters refer to the following terms:

Character	Term
a, b, A, B	Term or word in a query or document.
$c_1, c_2, \ldots c_m$	Root or concept node in query network.
$d_1, d_2, \ldots d_i$	Document node in a document network.
D	Number of documents to be selected or identified to result list.
f_i	Concept frequency in collection (frequency, or number, of documents in collection containing concept i).
f_{ij}	Frequency of concept i in document j.
f_{max}	Probable maximum frequency of documents in collection containing specific concept (maximum bound).
f_{min}	Probable minimum frequency of documents in collection containing specific concept (minimum bound).
g	Number of documents in collection between documents containing a representation (gaps).
I	Information need in query network.
i	Concept (an item of an information need).
idf_i	Inverse document frequency for concept i.
idf_{imax}	Probable maximum inverse document frequency for concept i.
idf_{min}	Probable minimum inverse document frequency for concept i.
j	Specific document (d_j).
max f_j	The maximum frequency for any term occurring in document j.
n_i	Number of documents in sample containing selected representation.
n_c	Number of documents in collection.
$P_1, P_2, \ldots P_n$	Parent nodes to child node Q.
$q_1, q_2, \ldots$	Query nodes in query network.
Q	Child node to parent nodes P.
$r_1, r_2, \ldots r_k$	Leaf or concept representation nodes in document network.
s_i	A calculated number equal to greater of x_i/n_i and sd.
sd	Standard deviation.
sq	Sum of squares of gaps g.
$t_1, t_2, \ldots t_j$	Interior text nodes in document network.
tf_{ij}	Probability estimate based on the frequency that concept i appears in document j (based on f_{ij}).
T	Number of terms in query.
V	Number of duplicate terms removed from query.
$w_1, w_2, \ldots w_n$	Term weights for parent nodes where w_g is maximum.
w_g	Maximum term weight for child node Q, $0 \leq w_g \leq 1$.
x_i	Number of documents in sample.

-continued

Character	Term
z	Standard critical value.
π	Parent Set (P_1, P_2, . . . P_n)

SUMMARY OF THE INVENTION

According to one aspect of the present invention the frequency of occurrence of a selected representation in a collection of documents is estimated by identifying the frequency of occurrence of the representation in a sample of documents selected from the collection. Probable maximum and probable minimum frequencies of occurrence of the representation in the entire collection are calculated, and the midpoint of the probable maximum and minimum frequencies is selected.

The estimated frequency of occurrence of the selected representation is set equal to the selected midpoint when the calculated difference between the probable maximum and minimum frequencies does not exceed a preselected limit. If the preselected limit is exceeded, the sample of documents is adjusted to include additional documents from the collection, the sampling and calculating being repeated until the calculated difference between the probable maximum and minimum frequencies is within the preselected limit.

The advantage provided by estimation of the frequency of representations such as phrases, synonyms, proximities and thesaurus classes is that the representations can be identified from the query itself and the frequencies can be accurately estimated without significantly affecting processing resources or the search results. Consequently, representations such as phrases, synonyms, proximities and thesaurus classes can be employed as representation concepts, even in large databases.

According to another aspect of the invention a sample is selected and the one document with the highest probability of meeting the information need defined by the query is identified from the sample of documents from the collection. In one form of the invention, a probability threshold is set equal to the probability that the selected document meets the information need. When a predetermined number of additional documents of the collection are identified as having a probability of meeting the information need which is greater than the probability threshold, the threshold is reset to the probability of the selected document with the lowest calculated probability. Thereafter, as documents with higher probabilities are identified, the documents with the lowest probabilities are correspondingly removed. Upon completion of the search, the predetermined number of documents identified as having the highest probabilities are retrieved, preferably in probability order.

In another form of the invention, instead of employing the probability of the document selected from the first sample as a probability threshold, successive samples are iteratively selected, each successive sample containing documents different from each previous sample. Up to a predetermined number of documents having the highest probabilities of meeting the information need are identified during each iteration, the documents being selected from a group consisting of the sample of documents selected for the respective iteration and the documents identified during the previous iteration. Preferably, the predetermined number is equal to the number of the respective iteration, so there are as many iterations as there are documents to be selected.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram representation of a Bayesian inference network with which the present invention is used.

FIG. **2** is a block diagram representation of a simplified Bayesian inference network as in FIG. **1**.

FIG. **3** is a block diagram of a computer system for carrying out the invention.

FIGS. **4A** and **4B**, taken together, are a flowchart and example illustrating the steps of creating a search query for a probabilistic network.

FIG. **5** is a flowchart and example of the steps for determining a key number for inclusion in the search query described in connection with FIG. **4**.

FIGS. **6A–6D** are block diagram representations of illustrating different techniques for handling phrases.

FIGS. **7A** and **7B**, taken together, are a detailed flowchart identifying the steps for calculating the estimated inverse document frequency for a specific concept according to the present invention.

FIG. **8** is a flowchart illustrating the manner by which partial phrases are handled in a document retrieval system.

FIG. **9** is a graph illustrating the principles of certain aspects of threshold estimating according to the present invention.

FIG. **10** is a detailed flowchart identifying the steps for setting probability thresholds and optimizing document retrieval according to the present invention.

FIG. **11** is a detailed flowchart illustrating the maximum score optimization techniques according to the present invention.

FIG. **12** is a detailed flowchart of the process for creating the query network for a probabilistic information retrieval network.

FIG. **13** is a detailed flowchart of the process for evaluating a document network used with the query network shown in FIG. **12**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The Probability Network

Inference probability networks employ a predictive probability scheme in which parent nodes provide support for their children. Thus, the degree to which belief exists in a proposition depends on the degree to which belief exists in the propositions which potentially caused it. This is distinct from a diagnostic probability scheme in which the children provide support for their parents, that is belief in the potential causes of a proposition increases with belief in the proposition. In either case, the propagation of probabilities through the network is done using information passed between adjacent nodes.

FIG. **1** illustrates a Bayesian inference network as described in the aforementioned Turtle et al. article. The Bayesian network shown in FIG. **1** is a directed, acyclic dependency graph in which nodes represent propositional variables or constraints and the arcs represent dependence relations between propositions. An arc between nodes represents that the parent node "causes" or implies the proposition represented by the child node. The child node contains a link matrix or tensor which specifies the probability that the child node is caused by any combination of the parent nodes. Where a node has multiple parents, the link matrix specifies the dependence of that child node on the set of

parents and characterizes the dependence relationship between the node and all nodes representing its potential causes. Thus, for all nodes there exists an estimate of the probability that the node takes on a value given any set of values for its parent nodes. If a node a has a set of parents $\pi a=\{p_1, \ldots p_n\}$, the estimated probabilities $P(a|p_1, \ldots p_n)$ are determined.

The inference network is graphically illustrated in FIG. 1 and consists of two component networks: a document network **10** and a query network **12**. The document network consists of document nodes d_1, d_2, . . . d_{t-1}, d_t, interior text representation nodes t_1, t_2, . . . t_{j-1}, t_j, and leaf nodes r_1, r_2, r_3, . . . r_k. The document nodes d correspond to abstract documents rather than their physical representations. The interior nodes t are text representation nodes which correspond to specific text representations within a document. The present invention will be described in connection with the text content of documents, but it is understood that the network can support document nodes with multiple children representing additional component types, such as audio, video, etc. Similarly, while a single text may be shared by more than one document, such as journal articles that appear in both serial issue and reprint collections, and parent/divisional patent specifications, the present invention shall be described in connection with a single text for each document. Therefore, for simplicity, the present invention shall assume a one-to-one correspondence between documents and texts.

The leaf nodes r are content representation nodes. There are several subsets of content representation nodes r_1, r_2, r_3, . . . r_k, each corresponding to a single representation technique which has been applied to the document texts. If a document collection has been indexed employing automatic phrase extraction and manually assigned index terms, then the set of representation nodes will consist of distinct subsets or content representation types with disjoint domains. For example, if the phrase "independent contractor" has been extracted and "independent contractor" has been manually assigned as an index term, then two content representation nodes with distinct meanings will be created, one corresponding to the event that "independent contractor" has been automatically extracted from the subset of the collection, and the other corresponding to the event that "independent contractor" has been manually assigned to a subset of the collection. As will become clear hereinafter, some concept representation nodes may be created based on the content of the query network.

Each document node has a prior probability associated with it that describes the probability of observing that document. The document node probability will be equal to 1/(collection size) and will be small for most document collections. Each text node contains a specification of its dependence upon its parent. By assumption, this dependence is complete (t_t is true) when its parent document is observed (d_t is true). Each representation node contains a specification of the conditional probability associated with the node given its set of parent text nodes. The representation node incorporates the effect of any indexing weights (for example, term frequency in each parent text) or term weights (inverse document frequency) associated with the concept.

The query network **12** is an "inverted" directed acyclic graph with a single node I which corresponds to an information need. The root nodes c_1, c_2, c_3, . . . c_m are the primitive concept nodes used to express the information requirement. A query concept node, c, contains the specification of the probabilistic dependence of the query concept on its set of parent representation content nodes, r. The query concept nodes c_1 . . . c_m define the mapping between the concepts used to represent the document collection and the concepts that make up the queries. A single concept node may have more than one parent representation node. For example, concept node c_2 may represent the query concept "independent contractor" and have as its parents representation nodes r_2 and r_3 which correspond to "independent contractor" as a phrase and as a manually assigned term.

Nodes q_1, q_2 are query nodes representing distinct query representations corresponding to the event that the individual query representation is satisfied. Each query node contains a specification of the query on the query concept it contains. The intermediate query nodes are used in those cases where multiple query representations express the information need I.

As shown in FIG. **1**, there is a one-to-one correspondence between document nodes, d, and text nodes, t. Consequently, the network representation of FIG. **1** may be diagrammatically reduced so that the document nodes d_1, d_2 , . . . d_{t-1}, d_t are parents to the representation nodes r_1, r_2, r_3, . . . r_k. In practice, it is possible to further reduce the network of FIG. **1** due to an assumed one-to-one correspondence between the representation nodes r_1, r_2, r_3, . . . r_k, and the concept nodes c_1, c_2, c_3, . . . C_m. The simplified inference network is illustrated in FIG. **2** and is more particularly described in the article by Turtle et al., "Efficient Probabilistic Inference for Text Retrieval," RIAO **91** Conference Proceedings, pp. 644–661, April, 1991 (Recherche d'Informaion Assistée par Ordinateur, Universitat Autónoma de Barcelona, Spain), which article is herein incorporated by reference.

As described above, each child node carries a probability that the child node is caused by the parent node. The estimates of the dependence of a child node Q on its set of parents, P_1, P_2, . . . P_n, are encoded using the following expressions:

$$bel_{or}(Q) = 1 - (1 - p_1) \cdot (1 - p_2) \cdot \ldots \cdot (1 - p_n) \quad \text{EQ 1}$$

$$bel_{and}(Q) = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_n \quad \text{EQ 2}$$

$$bel_{not}(Q) = 1 - p_1 \quad \text{EQ 3}$$

$$bel_{wsum}(Q) = \frac{(w_1 p_1 + w_2 p_2 + \ldots + w_n p_n) w_g}{w_1 + w_2 + w_2 + \ldots w_n} \quad \text{EQ 4}$$

where $P(P_1=\text{true})=p_1$, $P(P_2=\text{true})=p_2$, . . . $P(P_n=\text{true})=p_n$, w_1, W_2, . . . W_n are the term weights for each term P_1, P_2, . . . P^n, and w_g is the maximum probability that the child node can achieve, $0 \leqq W_g \leqq 1$.

As described above, all child nodes carry a probability that the child was caused by the identified parent nodes. The structure of document network **10** is not changed, except to add documents to the database. The document nodes d and text nodes t do not change for any given document once the document representation has been entered into document network **10**. Most representation nodes are created with the database and are dependent on the document content. Some representation nodes (representing phrases and the like) are created for the particular search being conducted and are dependent on the search query.

Query network **12**, on the other hand, changes for each input query defining a document request. Therefore, the concept nodes c of the search network are created with each search query and provide support to the query nodes q and the information need, node I (FIG. **1**).

Document searching can be accomplished by a document-based scan or a concept-based scan. A document-based scan is one wherein the text of each document is scanned to determine the likelihood that the document meets the information need, I. More particularly, the representation nodes

$r_1, r_2, r_3, \ldots r_k$ of a single document are evaluated with respect to the several query nodes q_1, q_2 to determine a probability that the document meets the information need. The top D-ranked documents are then selected as potential information need documents. The scan process reaches a point, for example after assigning a probability for more than D documents of a large document collection, that documents can be eliminated from the evaluation process after evaluating subsets of the representation nodes. More particularly, if a given document scores so low of a probability after only evaluating one or two representation nodes, determination can be made that even if the evaluation continued the document still would not score in the top D-ranked documents. Hence, most documents of a large collection are discarded from consideration without having all their representation nodes evaluated.

A concept-based scan is one wherein all documents containing a given representation node are evaluated. As the process continues through several representation nodes, a scorecard is maintained of the probabilities that each document meets the information need, I. More particularly, a single representation node r_i is evaluated for each document in the collection to assign an initial probability that the document meets the concept. The process continues through the several representation nodes with the probabilities being updated with each iteration. The top D-ranked documents are then selected as potential information need documents. If at some point in the process it can be determined that evaluation of additional representation concepts will not alter the ranking of the top D-ranked documents, the scan process can be terminated.

It can be appreciated that the representation nodes $r_1, r_2, r_3, \ldots r_k$ are nodes dependent on the content of the texts of the documents in the collection. Most representation nodes are created in the document database. Other representation nodes, namely those associated with phrases, synonyms and citations, are not manifest in any static physical embodiment and are created based on each search query. Because the user can define phrases and thesaurus relationships when creating the query, it is not possible to define all combinations in a static physical embodiment. For example, a query manifesting the concept "employee" may be represented by one or more of "actor", "agent", "attendant", "craftsman", "doer", "laborer", "maid", "servant", "smith", "technician" and "worker", to name a few. These various representation nodes may be created from the query node at the time of the search, such as through the use of thesauri and other tools to be described, as well as through databases. A query node q_1, q_2, etc. can be manifest in one or more representations.

The Search Query

The present invention will be described in connection with a database for searching legal documents, but it is to be understood the concepts of the invention may be applied to databases for searching other types or classes of documents. The invention will be described in connection with a specific search query as follows:

> "What is the liability of the United States under the Federal Tort Claims Act for injuries sustained by employees of an independent contractor working under contract with an agency of the United States government?"

The present invention is carried out through use of a computer system, such as illustrated in FIG. 3 comprising a computer 20 connected to an input/output terminal 22 and a read only memory (ROM) 24. ROM 24 may be any form of read only memory, such as a CD ROM, write protected magnetic disc or tape, or a ROM, PROM or EPROM chip encoded for the purposes described. Computer 20 may be a personal computer (PC) and may be optionally connected through modem 26, telephone communication network 28 and modem 30 to a central computer 32 having a memory 34. In one form of the invention, the document network 10 and the document database containing the texts of documents represented by the document network are contained in the central computer 32 and its associated memory 34. Alternatively, the entire network and database may be resident in the memory of personal computer 20 and ROM 24. In a legal database and document information retrieval network the documents may comprise, for example, decisions and orders of courts and government agencies, rules, statutes and other documents reflecting legal precedent. By maintaining the document database and document network at a central location, legal researchers may input documents into the document database in a uniform manner. Thus, there may be a plurality of computers 20, each having individual ROMs 24 and input/output devices 22, the computers 20 being linked to central computer 32 in a time-sharing mode. The search query is developed by each individual user or researcher and input via the respective input/output terminal 22. For example, input/output terminal 22 may comprise the input keyboard and display unit of PC computer 20 and may include a printer for printing the display and/or document texts.

ROM 24 contains a database containing phrases unique to the specific profession to which the documents being searched are related. In a legal search and retrieval system as described herein, the database on ROM 24 contains stemmed phrases from common legal sources such as *Black's* or *Statsky's Law Dictionary*, as well as common names for statutes, regulations and government agencies. ROM 24 may also contain a database of basic and extended stopwords comprising words of indefinite direction which may be ignored for purposes of developing the concept nodes of the search query. For example, basic stopwords included in the database on ROM 24 includes indefinite articles such as "a", "an", "the", etc. Extended stopwords include prepositions, such as "of", "under", "above", "for", "with", etc., indefinite verbs such as "is", "are", "be", etc. and indefinite adverbs such as "what", "why", "who", etc. The database on ROM 24 may also include a topic and key database such as the numerical keys associated with the well-known West Key Digest system.

FIGS. 4A and 4B are a flow diagram illustrating the process steps and the operation on the example given above in the development of the concept nodes c. The natural language query is provided by input through input terminal 22 to computer 20. In the example shown in FIG. 4, the natural language input query is:

"What is the liability of the United States under the Federal Tort Claims Act for injuries sustained by employees of an independent contractor working under contract with an agency of the United States government? "

By way of example, a corresponding WESTLAW Boolean query might be:

"UNITED STATES" U.S. GOVERNMENT (FEDERAL /2 GOVERNMENT) /P TORT/2 CLAIM/P INJUR! /P EMPLOYEE WORKER CREWMAN CREWMEMBER /P INDEPENDENT/2 CONTRACTOR.

As shown in FIG. 4A, the natural language query shown in block 40 is inputted at step 50 to computer 20 via input/output terminal 22. The individual words of the natural language query are parsed into a list of words at step 50, and at step 54 each word is compared to the basic stopwords of the database in ROM 24. At step 54, the basic stopwords

such as "the" are removed from the list. The extended stopwords are retained for phrase recognition and remaining extended stopwords will be removed after phrase recognition, described below.

At step **56**, the remaining words are stemmed to reduce each word to its correct morphological root. One software routine for stemming the words is based on that described by Porter "An Algorithm for Suffix Stripping", Program, Vol. 14, pp 130–137 (1980). As a result of step **56** a list of words is developed as shown in block **42**, the list comprising the stems of all words in the query, except the basic stopwords.

Phrases

Previous systems recognized linguistic structure (for example, phrases) by statistical or syntactic techniques. Phrases are recognized using statistical techniques based on the occurrence of phrases in the document collection itself; thus, proximity, co-occurrence, etc. were used. Phrases are recognized using syntactic techniques based on the word/term structure and grammatical rules, rather than statistically. Thus, the phrase "independent contractor" could be recognized statistically by the proximity of the two words and the prior knowledge that the two words often appeared together in documents. The same term could be recognized syntactically by noting the adjective form "independent" and the noun form "contractor" and matching the words using noun phrase grammatical rules. (Manual selection systems have also been used wherein the researcher manually recognizes a phrase during input.)

Previous inference networks employed a two-term logical AND modeled as the product of the beliefs for the individual terms. Beliefs (probabilities) lie in the range between 0 and 1, with 0 representing certainty that the proposition is false and 1 representing certainty that the proposition is true. The belief assigned to a phrase is ordinarily lower than that assigned to either component term. However, experiments reveal that the presence of phrases represents a belief higher than the belief associated with either component term. Consequently, separately identifying phrases as independent representation nodes significantly increases the performance of the information retrieval system. However, single terms of an original query are retained because many of the concepts contained in the original query are not described by phrases. Experimentation has suggested that eliminating single terms significantly degrades retrieval performance even though not all single terms from an original query are required for effective retrieval.

As previously described, the phrase relationships in the search query are recognized by domain-knowledge based techniques (e.g., the phrase database), and by syntactic relationships. The primary reason to solely select syntactical and domain-based phrases for purposes of the query network is to reduce user involvement in identifying phrases for purposes of creating a query.

An example of a domain-knowledge database is a database containing phrases from a professional dictionary. This type of phrase handling is particularly suitable for professional information retrieval where specialized phrases are often employed.

At step **58** in FIG. 4B, computer **20** returns to the database in ROM **24** to determine the presence of phrases within the parsed and stemmed list **42**. The phrase database in ROM **24** comprises professional, domain-specific phrases (such as from *Black's Law Dictionary*) which have been stemmed in accordance with the same procedure for stemming the words of a search query. Computer **20** compares the first and second words of list **42** to the database of phrases in ROM **24** to find any phrase having at least those two words as the first words of a phrase. Thus, comparing the first two terms "WHAT" and "IS" to the database of phrases (such as *Black's Law Dictionary*), no match is found. Thus, as shown in block **44**, "WHAT" is retained for the search query. The next two words "IS" and "LIABL" are compared to the database of phrases and no phrase is found. When "UNITE" and STATE" are compared to the database, a phrase match is found. The next word "FEDERAL" is then compared to the database to determine if it corresponds to the third word of any phrase commencing with "UNITE STATE". In this case no phrase is found, so both "UNITE" and "STATE" are removed from the list **44** and substituted with a phrase representing the term "UNITE STATE". When the terms "FEDERAL" and "TORT" are compared to the database a match is found to phrases in the database. The third and fourth words "CLAIM" and "ACT" also compare to at least one phrase commencing with "FEDERAL" and "TORT". Consequently, each of the terms "FEDERAL", "TORT", "CLAIM" and "ACT" are substituted with the phrase "FEDERAL TORT CLAIM ACT". (As explained below, if a word is found to be included in a successive phrases, the common word would be assigned to the longer phrase, if they have an unequal number of terms, or to the first phrase of the succession, if the number of terms in the phrases are equal.) The process continues to substitute phrases from the database for sequences of stemmed words from the parsed list **42**, thereby deriving the list **44**.

The phrase lookup is accomplished one word at a time. The current word and next word are concatenated and used as a key for the phrase database query. If a record with the key is found, the possible phrases stored under this key are compared to the next word(s) of the query. As each phrase is found, a record of the displacement and length of each found phrase is recorded.

The extended stopwords are included in the phrase matching technique because the phrases themselves contain such stopwords. For example, phrases like "doctrine of equivalents" and "tenancy at will" contain prepositions which are stopwords.

As indicated above, once successive terms have been identified as a phrase, the individual terms do not appear in the query shown at block **44** in FIG. 4B. In rare cases two phrases might seemingly overlap (i.e., share one or more of the same words). In such a case, the common word is not repeated for each phrase, but instead preference in the overlap is accorded to the longer phrase. For example, if a natural language search query contained "... tenancy at will, the power of which . . . ", the parsed and stemmed list (with basic stopwords removed) would appear as: "tenan", "at", "will", "power", "of", "which". The database could identify two possible phrases: "tenan at will" and "will power" with "will" in both phrases. As will be explained below, preference is accorded to the longest possible phrase, so the identified phrase will be "tenan at will".

With the phrases identified, as at **44**, the remaining extended stopwords ("what", "is", "of", "under", "for", "by", "with") are removed at step **62**, and any duplicate terms are removed at step **64**, to be described in greater detail below. The result is the final query shown at block **46** in FIG. 4B.

Citations

Case citations, U.S. Code citations and citations to the Code of Federal Regulations (CFR) are handled as exact terms. Other citations, including subsection citations, are handled syntactically using word-level proximity as single terms or query nodes comprising numeric tokens. For example, a citation to Volume **78** Columbia Law Review

page 1587 is encoded as 78 +4 1587 (meaning 78 within four words of 1587), and the citation to 17 U.S.C. 106A(e)(1) is encoded as 17+2 106A(e)(1). To encompass most citations, it is preferred to encode all citations as within five words. Hence, the above two citations will be encoded as 78+5 1587 and 17+5 106A(e) (1).

Hyphenations

Hyphenated terms in search queries are handled in much the same manner as citations. The hyphen is removed and the component words are searched using an adjacency operation which finds all adjacent occurrences of the component words.

Synonyms

Synonyms comprise equivalent words and misspellings and are created from a predefined database stored in ROM 24 (FIG. 3). Examples of equivalencies include 2d/2nd/second whereas examples of misspellings include habeas/habeus. Where a search query includes a word having a synonym, a new representation node r (FIG. 2) is created for each synonym. However, the weight associated with the node is based on the frequency of the entire class of nodes comprising all synonyms, rather than any one term of the class.

Duplicate terms

Where a single word, term or phrase occurs more than once in a query, the word, term or phrase is evaluated only once. After the word, term or phrase has been processed for phrase identification as heretofore described, the duplicate word, term or phrase is simply dropped from the search query. As will be explained hereinafter, the component probability score for each document containing a term duplicated in the query is multiplied by the query frequency, and the query normalization factor is increased by that frequency. Thus, the effect is that the duplicated term is evaluated multiple times as dictated by the query, but in a computationally simpler manner.

Thesaurus Classes

Thesauri are employed to identify words of similar or related meaning, as opposed to synonyms having identical meaning. The thesauri are used to suggest broader, narrower and related terms to the researcher for inclusion in the search query. These relationships can be drawn from the machine readable dictionaries (such as *Black's Law Dictionary*) encoded in databases, or from manually recorded domain knowledge.

Document Retrieval

One feature of probabilistic information retrieval systems is that the documents in the document collection are ranked in accordance with the probability that the document meets the information need identified in the query. This permits selection of a predetermined number of documents having the highest probabilities for identification and retrieval. For a given information need, for example, it may be desirable to retrieve 20 documents from a document collection of 500,000 documents. A probabilistic information retrieval network can identify for retrieval the 20 documents having the highest probability of meeting the information need.

Phrases, synonyms, proximities and thesaurus classes are not separately permanently identified in the document network. Instead, the representation nodes in the document network are created for the phrase, synonym, proximity or thesaurus class by those concept nodes (FIG. 1) which themselves are a function of the phrase or term in the query.

FIGS. 6A–6D illustrate different treatments of phrases in the document network of an inference network. Representation concepts r_1 and r_2 shown in FIGS. 6A–6D correspond to two words in the text of document d_m. Representation concept r_3 corresponds to the phrase in the text consisting of the two words. Q represents the query. For example, r_1 and r_2 may correspond to the occurrence of the terms "independent" and "contractor", respectively, while r_3 corresponds to the occurrence of the phrase "independent contractor". In the model illustrated in FIG. 6A (which is the preferred model), the phrase is treated as a separate representation concept, independent of the concepts corresponding to the component words. The belief in the phrase concept can be estimated using evidence about component words and the relationship between them, including linguistic relationships. The presence of the query phrase concept in the document increases the probability that the document satisfies the query (or information need). The model of FIG. 6B illustrates the case where the belief in the phrase concept depends on the belief in the concepts corresponding to the two component words. FIG. 6C illustrates a term dependence model where the phrase is not represented as a separate concept, but as a dependence between the concepts corresponding to the component words. A document that contains both words will more likely satisfy the query associated with the phrase due to the increase belief coming from the component words themselves. However, experimentation has revealed that the model of FIG. 6C is less appropriate for phrases and more appropriate for thesauri and synonyms. In FIG. 6D belief in the phrase concept is established from evidence from the document text itself, whereas belief in the concepts representing component words are derived from belief in the phrase itself. The model of FIG. 6D makes explicit the conditional dependence between the component concepts and addresses the practice of some authors that all component words of a phrase might not always be used in the text representation of a document. For the present purposes, it is preferred that document network 10 employ the phrase model of FIG. 6A so that the representation concepts for the phrases are independent of the corresponding words. Hence, a match between the concept node of a search query and the concept node of a documentation representation is more likely to occur where the search query contains only the phrase, and not the component words. It is understood that the other models (FIGS. 6B–6D) could be employed with varying results.

Thus far, there has been described techniques for obtaining lists containing single words, phrases, proximity terms (hyphenations and citations) and key numbers. These elements represent the basic concept nodes contained in the query. The phrases, hyphenations and citations create representation nodes of the document network. Computer 20 (FIG. 3) forwards the search query to computer 32, which determines the probability that a document containing some subset of these concepts matches the original query. For each single document, the individual concepts represented by each single word, phrase, proximity term, and key number of the query are treated as independent evidence of the probability that the document meets the information need, I. The probability for each concept is determined separately and combined with the other probabilities to form an overall probability estimate.

The probabilities for individual concepts are based on the frequency with which a concept occurs in document j (tf_{ij}) and the frequency (f_i) with which documents containing the concept (i) occur in the entire collection. The collection frequency may also be expressed as an inverse document frequency (idf*i*). The inference network operates on two basic premises:

A concept that occurs frequently in a document (a large tf_{ij}) is more likely to be a good descriptor of that document's content, and

A concept that occurs infrequently in the collection (a large idf_i) is more likely to be a good discriminator than a concept that occurs in many documents.

It can be shown that the probability, $P(c_i|d_j)$ that concept c_i is a "correct" descriptor for document d_j may be represented as

$$P(c_i|d_j) = 0.4 + 0.6 \cdot idf_i \cdot tf_{ij}, \quad \text{EQ 5}$$

where

$$tf_{ij} = 0.5 + 0.5 \cdot \frac{\log (f_{ij} + 1)}{\log max f_j} \quad \text{EQ 6}$$

and

$$idf_i = \frac{\log \frac{n_c}{f_i}}{\log n_c} \quad \text{EQ 7}$$

if f_{ij} is less than max f_j, where n_c is the number of documents in the collection, f_{ij} is the frequency of concept i in document j, f_i is the frequency of documents in the collection containing term i (i.e., the number of documents in which term i occurs), and max f_j is the maximum frequency for any term occurring in document j. If f_{ij} is not less than max f_j, then $tf_{ij}$is set to 1.

Most document networks for search and retrieval are represented by a word index containing words from the documents to be matched to query terms. In Boolean networks, relationships were determined from the word index and offset data therein to locate documents meeting the logical criteria of the query. The present invention employs a probabilistic network in which the same database and word index may be employed to calculate the probabilities set forth in Equation 5 for many of the query concepts. The number of documents in the collection, n_c, is known from the document addresses associated with words in the word index. To calculate f_i, the number of documents in the collection containing concept i is determined by locating and counting the addresses of all documents in the database containing the concept. More particularly, the document addresses associated with each word in the word index corresponding to the concept are compared to remove duplicate addresses and the remaining number of document addresses is summed. The resulting sum is f_i. The frequency or number of times, f_{ij}, that concept i appears in document j can be calculated from the number of offset codes for the word (and its synonyms) associated with the document. Hence, the terms idf_i and tf_{ij} can be calculated, thereby leading to the probability factor, $P(c_i|d_j)$, for the concept for the document in accordance with Equation 5. However, this technique is useful only for those concepts whose concept frequency is represented in the word index. Certain concepts, such as phrases, are not ordinarily so represented, so it is an aspect of the present invention to provide a technique to estimate the representation concept frequency for such concepts.

Representation Concept Frequency Estimation

The inverse document frequency (idf_i) is predetermined for each representation concept in the document collection, except certain representations such as phrases, synonyms, proximities and thesaurus classes. For phrases, synonyms, proximities and thesaurus classes, the inverse document frequency is computed for each search. Identifying the inverse document frequency for a given phrase, synonym, proximity or thesaurus class requires processing through each document in the collection. In small collections, the computation of the inverse document frequency of a phrase, synonym, proximity, or thesaurus class may be performed without significant difficulty by examination of the word index to determine f_i, n_c and f_{ij} as described above. Hence, the inverse document frequency for the phrase may be calculated using equation 7. However, in the case of large collections (of the order of 500,000 documents), computation of the inverse document frequency for a phrase, synonym, proximity or thesaurus class representation requires significant processing, if all documents containing a query concept are to be examined. Moreover, in many circumstances the computation may lead to a result which is too insignificant to affect the ranking.

Consider, for example, a synonym class containing terms A and B where term A occurs in 10,000 documents in the collection of 500,000 documents and term B occurs in 10 documents. The frequency of the synonym class lies in the range of 10,000 to 10,010, resulting in a frequency difference of 10 documents in 10,010 or about 0.1%. Consequently, the range of the inverse document frequency, idf_i, lies between about 0.02000 and 0.02002, which is too small to significantly affect the result ranking. However, if term A appears in 10,000 documents and term B appears in 4,000 documents, the frequency is in the range of 10,000 and 14,000 , leaving a 28.6% frequency difference and a range of document inverse frequencies between 0.02000 and 0.02800, which is significant.

One aspect of the present invention concerns the estimation of the inverse document frequency for a selected representation, such as a phrase, proximity, synonym or thesaurus class. More particularly, the representation frequency is estimated from a sample of the collection with sufficient accuracy, while avoiding extended computational resources in the evaluation of the entire collection. A sample of a plurality of documents is selected from the collection, and the representations in the sample documents are processed to identify the frequency that the selected representation occurs in the sample. Specifically, the "gaps," or the number of documents (g) occurring between occurrences of documents containing the selected representation, are identified, and the sum of the squares of the gaps (sq) are employed to estimate the correct representation frequency. The gaps are identified from the successive addresses of documents containing the concept as determined from the word index of the document database. The sequence of observed gaps are employed to estimate the maximum and minimum bounds (f_{max} and f_{min}) of the true frequency within a preselected error rate. The frequency bounds are employed to compute the range of the probable inverse document frequency. When that range becomes sufficiently narrow as to insignificantly affect the result ranking, the midpoint of the frequency range is selected as the estimated frequency of occurrence of the selected representation.

After computing the frequency bounds for the given sample, if the difference between the bounds is too large that the selection of the midpoint as the estimated frequency of occurrence is likely to affect the result ranking, the sample is enlarged to include additional documents, and the frequency bounds are again computed. Ordinarily, mean and variance estimations are computed on the basis that each sample is independent, but in the present case the samples may not be independent because samples are taken sequentially, rather than randomly. To adjust for possible non-random sampling, the variation for the frequency bounds is estimated in two ways: first based on random sampling, and second based on gaps (numbers of documents found between documents containing the representation). The probable maximum frequency, f_{max}, and the probable minimum frequency, f_{min}, are computed in accordance with the following algorithms:

$$f_{max} = n_i + \frac{n_i(n_c - x_i)}{x_i - zs_i\sqrt{n_i}} + z\sqrt{\frac{n_i(n_c - x_i)}{x_i}} \quad \text{EQ 8}$$

and

$$f_{min} = n_i + \frac{n_i(n_c - x_i)}{x_i + zs_i\sqrt{n_i}} - z\sqrt{\frac{n_i(n_c - x_i)}{x_i}} \quad \text{EQ 9}$$

where

n_i is the number of documents (or gaps between documents) in the sample containing the selected representation,

n_c is the number of documents in the collection,

x_i is the number of documents in the sample,

s_i is the greater of x_i/n_i or sd of the n_i gaps, and

z is the standard critical value for normal distribution for a preselected reliability,

and where sd is the standard deviation and is represented by

$$sd = \sqrt{\frac{sq}{n_i} - \left(\frac{x_i}{n_i}\right)^2} \quad \text{EQ 10}$$

where sq is the sum of the squares of the gaps, or the sum of the squares of the numbers of documents found between documents containing the representation.

It is preferred that the reliability of the estimation be within 0.95 (i.e., the maximum error rate should not exceed 5%). It can be shown that the standard critical value (z) for a normal distribution of the documents of the collection, within a 0.95 reliability, is 2.8070.

There are several constraints on the calculation of f_{max} and f_{min}. First, if f_{min} is smaller than the a priori minimum, then f_{min} is set equal to the a priori minimum, and if f_{max} is greater than the a priori maximum, then f_{max} is set equal to the a priori maximum. To illustrate the a priori minimums and maximums, assume a synonym class containing terms A and B where term A appears in 10,000 documents and term B appears in 4,000 documents. Terms A and B could appear in the same or overlapping documents, meaning that term B could appear in as many as 4,000 documents with term A. Conversely, term B might appear in documents exclusive of term A. Consequently, although the actual occurrences of the synonym class is unknown, the synonym class appears in the range of 10,000 to 14,000 documents. Hence, an a priori minimum number of occurrences can be established at 10,000 (the number of occurrences of the most common term A), and an a priori maximum number of occurrences can be established at 14,000 (the sum of occurrences of both terms A and B). Similarly, in the case of a phrase containing two terms A and B (such as "independent contractor"), if A appears in 10,000 documents and B appears in 4,000 documents, an a priori maximum exists of 4,000 (the number of occurrences of the least common term B) because that is the maximum that the two terms could appear together.

Hence, the a priori maximums and minimums are derived from the pre-identified frequencies f_i of individual terms (which form or are part of the concept) in the collection, and the type of concept (synonym, phrase, thesaurus or proximity).

Another constraint concerning the calculation of f_{min} is that if the calculated f_{min} is smaller than n_i (the number of documents in the sample containing the representation), f_{min} is set equal to n_i. Likewise, if the calculated f_{max} is smaller than zero or is less than n_i, f_{max} is set equal to $n_i + (n_c - x_i)$ (the number of documents in the sample containing the representation plus the number of documents of the collection yet to be considered).

The number of documents x_i in the sample necessary to estimate the frequency of the selected representation is increased until the difference between the inverse document frequencies of the maximum and minimum bounds is smaller than some prescribed amount.

While the specific limit of the difference between the maximum and minimum inverse document frequencies is heuristic, it has been found that when the range of frequency values between f_{max} and f_{min} is so small that further refinement would not significantly alter the ranking of the ultimately selected documents, further computation of an estimated probable frequency for the selected representation may be halted. For purposes of the present invention, an inverse document frequency (idf_i) difference of 0.05 or less as an empirically selected stopping point, provides good results. The estimated inverse document frequency for the selected representation is thereupon selected at the mean between the maximum and minimum bounds. If the maximum and minimum bounds are accurate, they would each be located at a maximum error of 0.025 which is deemed acceptable for the present purposes. In practice, the correct frequency error is usually smaller than 0.025 because the correct frequency tends to lie in the center of the estimated range more often than near either the maximum or minimum bound. Tests have indicated that the average error for the estimated frequency for the selected representation is about 0.01.

FIGS. 7A and 7B, taken together, comprise a detailed flowchart illustrating the steps of estimating the frequency of a selected concept, such as a phrase, synonym, proximity or thesaurus class. The process illustrated in FIG. 7A and 7B is carried out by a computer, which calculates the probable maximum and minimum frequencies f_{max} and f_{min} shown in Equations 8 and 9 and calculates the estimated inverse document frequency, idf_i, for the selected concept.

At step **70**, the number of documents in the sample (x_i), the number of documents in the sample containing the selected representation (n_i), the gap size (g), and the sum of the squares of the gaps (sq), are each initialized to 0. At step **72**, 1 is added to x_i and at step **74** the increased x_i is compared to n_c, the number of documents in the entire collection. If x_i is smaller than n_c, the first document j is examined at step **76** to determine whether or not concept i appears in the document. If the concept does not appear in the first document, 1 is added to g at step **78** and the sequence loops back through point **80** to increment x_i by 1. The process continues to loop until a document is identified containing concept i at step **76**. By that point, the value of g has been incremented and is equal to the number of documents not containing concept i since identifying the previous document containing concept i. At step **82**, n_i is incremented by 1, and at step **84** g^2 is calculated and is added to sq at step **86**. At step **88** g is reset to 0.

To conserve computing resources, it is preferred that f_{max} and f_{min} not be calculated each time a document is located containing concept i. Instead, it is preferred that a decision be made at step **90** which inhibits calculation of f_{max} and f_{min} until after only a predetermined number of documents containing the concept are identified. This has two effects: first, it conserves computing resources, and second, it permits use of the actual inverse document frequency (idf_i) for those concepts not appearing often in the collection. More particularly, it is preferred that a fixed number of documents, such as 25, be found containing concept i between each calculation of f_{max} and f_{min}. Thus, at step 90 n_i is divided by 25 and if the result is a whole number (indicating that n_i is 25, 50, 75, etc.), then the process continues through steps **92**,

94 and 96 to calculate f_{max} and f_{min}. On the other hand, if n_i is not equal to 25, 50, 75, etc., the process loops back through point 80 to continue to identify concept i in additional documents.

At step 92, x_i/n_i and sd are calculated, sd being calculated in accordance with equation 10. At step 94, s_i is set to the greater of x_i/n_i or sd. At step 96, f_{max} and f_{min} are calculated.

It should be noted that g is the size of the gap or the number of successive documents not containing the concept between documents that do contain the concept. Thus, g is incremented at step 78 for each document not containing the concept and is reset at step 88 upon finding a document which does contain the concept. Term sq calculated at step 86 is the sum of the squares of the gaps g.

After the maximum and minimum estimated bounds, f_{max} and f_{min}, are computed, maximum and minimum inverse document frequencies for the concept, idf_{imax} and idf_{imin}, are calculated at step 98. At step 100, if idf_{imin} is within 0.05 of idf_{imax}, the mean frequency f_{mean} is computed from f_{max} and f_{min} at step 102, and the estimated inverse document frequency, idf_i, is computed at step 104 for the concept. As shown at step 100, if the range between the maximum and minimum inverse document frequencies is greater than 0.05, the process loops back to point 80 to expand the sample and the number of documents until the bounds of the estimates are within 0.05 at step 100 or until the entire collection has been examined ($x_i=n_c$) at step 74.

As indicated above, it is possible that the entire collection could be examined before determining an estimated inverse document frequency for the selected concept. This might occur, for example, where a concept very rarely appears in the documents. In such a case, at step 74, the computer determines that the number of documents in the sample (x_i) is equal to the number of documents in the collection (n_c), in which case the actual inverse document frequency for the concept is computed at step 106.

Partial Concepts (Phrases and Proximities)

As shown by Equation 4, the probability is computed for each concept/document pair, and the probabilities are summed. The result is normalized by the number of concepts in the query to determine the overall probability estimate that the document satisfies the information requirement set forth in the query.

Phrases are treated in a manner similar to proximity terms, except that a document which does not contain the full phrase receives a partial score for a partial phrase. For example, if a query contains the phrase "FEDERAL TORT CLAIMS ACT" and a document contains the phrase "tort claims" but not "Federal Tort Claims Act", the document will receive a score based on the frequency distribution associated with "TORT CLAIMS". FIG. 8 is a flow diagram illustrating the process of handling partial matches. As shown at step 120, the full phrase is evaluated against the collection as heretofore described. The inverse document frequency (idf_i) is determined for the full phrase (step 122), and if idf_i is greater than a predetermined threshold (e.g., 0.3) the maximum belief achieved for any subphrase or single term is selected as the belief for the partial phrase (step 124). If idf_i is smaller or equal to the threshold value (0.3), the preselected default belief (0.4) is assigned to the documents containing the partial phrase (step 126).

Since the frequency of "TORT CLAIMS" must equal or exceed that of the longer phrase, the probability estimate for the partial phrase would generally be lower than that assigned to documents containing the complete phrase. For phrases which occur extremely often (for example, where idf_i is less than 0.3) it is preferred to dispense with the partial matching strategy, and treat the phrase as a pure proximity term by assigning the default belief (0.4) to all documents containing the partial phrase but not the full phrase (step 126). For phrases which appear less often (where idf_i is greater than 0.3), the maximum belief achieved by any single word of the partial phrase is assigned to the belief for the partial phrase.

As previously explained, duplicate terms are purged from the search query. However, where duplicate terms appear in the search query, the component probability score for each document containing the term is multiplied by the query frequency. For example, if a document contains a term which appears twice in a natural language query receives a component probability of 0.425, the probability score is multiplied by 2 (to 0.850) for that term. When the probabilities are summed and normalized as described above, the normalization factor is increased to reflect the frequency of the duplicated term (increased by 1 in this example). Thus, the duplicated term is treated as if it had been evaluated multiple times as dictated by the query, but in a computationally simpler manner.

As described above, the probability estimates for each document/concept pair are summed and the result is normalized by the number of concepts in the query. For the example given in FIG. 4 the search query shown in block 46 employs eleven concepts, so the total probability for each document will be divided by 11 to determine the overall probability that the given document meets the overall query. For example, assume for a given document that the eleven probabilities are:

0.400	0.430	0.466
0.543	0.436	0.433
0.512	0.400	0.481
0.460	0.472	

The overall probability is the sum of the individual probabilities (5.033) divided by the number of concepts (11) for a total probability of 0.458. This indicates a probability of 0.458 that the document meets the full query shown in block 40 in FIG. 4. The probability is determined for each document represented in the database, whereupon they are ranked in accordance with the value of the probability estimate to identify the top D documents. The ranking or identification is provided by computer 32 (FIG. 3) to computer 20 for display and/or printout at output terminal 22. Additionally, the document texts may be downloaded from computer 32 to computer 20 for display and/or printout at output terminal 22.

Probability Thresholds

As previously described, the probabilistic document retrieval system retrieves a predetermined number (D) of documents having the highest probability of meeting the information need set forth in the query. These probabilities are identified by the normalized sum of the probabilities of each representation in the document matching the concept in the query. Significant processor resources are required to compute these probabilities for each document in a large document database, for example about 500,000 documents or more. To reduce processing resources, it is desirable to limit probability computations to a reasonable number.

One technique to reduce processing resources is to employ a probability threshold against which the probabilities of documents are compared to determine whether or not the probability of a given document meets or exceeds the threshold. For example, in a document retrieval network designed to retrieve 10 documents, the probability threshold

may be set equal to the probability of the lowest ranked document of 10 selected documents. To identify 10 documents from a database of 500,000 documents, the first 10 documents of the database are listed to a result list (making the initial ranking of the top 10). A probability threshold is set equal to the probability of the lowest-ranked document of the first 10 selected documents. The probability of the 11th document is computed and compared against the probability threshold. If the probability of the 11th document exceeds that lowest ranked document of the original 10, the 11th document is entered into the result list of 10 selected documents and the prior lowest ranked document is removed. A new probability threshold is set to the probability of the new lowest ranked document of the original 10 selected documents. Hence, the probability threshold is a "running" threshold, constantly updated and increased in value as additional documents are identified which exceed the previous threshold.

It will be appreciated that at some point in the document identification process, the threshold becomes so high that many documents may be discarded from consideration after consideration of only a few of the representation probabilities. Assume, for example, a query containing eleven concepts and a probability threshold of 0.8965 (well into the document identification process). For a document to meet the threshold, it must have a minimum sum of individual probabilities of 9.8615 (11×0.8965). Under such circumstances, a low representation probability amongst the first few representations may result in a mathematical impossibility of meeting the threshold. For example, if the first two representations of a document have probabilities of 0.311 and 0.400, giving a sum of 0.711, it will not be possible for that document to make the result list of 10. Even if the representation probabilities matching the other nine concepts each had a probability of 1.0, the maximum sum of probabilities would be 9.711 which is normalized to a maximum probability of 0.8828, below the probability threshold. Consequently, it is unnecessary to calculate the additional representation probabilities for the document or to further process the document's probabilities.

It can be appreciated from the foregoing that comparing the document's probabilities against the threshold can provide a significant savings in processing resources.

While the foregoing probability thresholds provide significant savings in processing resources, particularly well into the search, very little savings is realized at the early stages of the search. FIG. **9** is a graph illustrating a threshold setting technique as described above. The process commences with a probability threshold of zero, following curve **130**. When the predetermined number of documents D are initially identified, the initial threshold is established as the lowest probability of the initial 10 documents, and subsequent documents are compared against the threshold. As additional documents are processed and the threshold value increases, it can be appreciated from FIG. **9** that the threshold value follows curve **130** approaching maximum threshold level **132**. It can be shown that the documents requiring examination against the probability is high at the early stages of the process and decreases as the process advances. Hence, the area of the graph of Figure **9** above the curve of line **130** is representative of the number of documents requiring processing and is representative of the required processing resources.

One feature of the present invention resides in the early estimations of the probability threshold for documents meeting the information need of the query. More particularly, by selecting a sample of documents and setting the initial probability threshold as equal to the probability of the document in the sample having the highest probability, an initial threshold may be established against which further documents may be compared as previously described. This "running start" is shown in FIG. **9** as the initial threshold for the process.

As the search continues through the collection, fewer documents have their probabilities scored and the probability threshold increases. Hence, document selection follows curve **134** in FIG. **9**. The establishment of an initial threshold as described, results in a smaller area above line **134**; the shaded area **136** represents a reduction in processing resources required for conducting the search.

It can be statistically shown that a document retrieval system, seeking to retrieve 10 documents meeting an information need defined by a query from a document collection of 500,000 documents, will, with a 5% maximum probable error rate, find one document in the first 309 documents, two documents in the first 11,095 documents, three documents in the first 25,070 documents, and so on in accordance with the following Table I:

TABLE I

Sequence	Limit (D)
309	1
11,095	2
25,070	3
48,843	4
80,269	5
118,159	6
161,889	7
211,278	8
266,579	9
500,000	10

The software algorithm for selecting the sequence of numbers for Table I is set forth below, where cs is the collection size (equal to n_c, the number of documents in the collection), gs is the goal size (equal to D, the number of documents to be selected or identified) and me is the maximum error sought. For Table I, cs is 500,000 , gs is 10 and me is 0.05.

SOFTWARE ALGORITHM

```
me = me + ((gs − 1) * 100)
conf = 1.0 − me
p    = gs + cs
lowi = (−log(conf)) + p        (natural log)
IF lowi = 0 THEN table(1) = lowi + 1
            ELSE table(1) = lowi
DO (j = 1 to (gs − 2))
      lowi = lowi + 1
      oldhi = cs − 1
      WHILE ((oldhi − lowi) <> 1)
            highi = ((lowi + oldhi − 1) + 2) + 1
            lambda = highi * p
            term = exp(−lambda)
            sum = term
            DO i = 1 TO j
                  term = term * (lambda + i)
                  sum = sum + term
            ENDDO
            IF sum > conf THEN lowi = highi
                        ELSE oldhi = highi
      ENDWHILE
      table (j+1) = lowi
ENDDO
table(gs) = cs
```

The forgoing software algorithm and Table I are employed to statistically optimize the probable document

distribution in the collection, and identifies one document to the result list during the first iteration, two documents to the result list during the second iteration, etc. until the final selection of ten documents are entered to the result list during the tenth iteration. During each iteration, a new sample of documents is selected from the collection, each sample being distinct from every other sample. Thus, referring to Table I, the first sample comprises documents 1 through 309, the second sample comprises documents 310 through 11095, the third sample comprises documents 11096 through 25070, etc. During the first iteration, the one document having the highest probability of meeting the information need defined by the query is selected from documents 1 through 309. During the second iteration, two documents having the two highest probabilities are selected from the group consisting of the sample of documents (documents 310 through 11095) plus the one document selected from the previous iteration. During the third iteration, three documents having the three highest probabilities are selected from the group consisting of documents 11096 through 25070 plus the two documents selected during the second iteration. The process continues through all iterations (10 in the example) to identify the predetermined number D of documents (10 in the example).

It is evident from the foregoing that if a given sample, such as the third sample, has two documents having probabilities which exceed the lowest of the previously selected documents, one previously selected document will be removed from the selection list. The ultimately selected documents, being ten in number, are not necessarily selected one from each of the ten samples. Instead, the selected documents are those ten documents having the highest probability of meeting the information need defined by the query, within a given error, such as 5%. While the above software algorithm sets forth the sample selection technique for any given number of documents to be identified, the above Table I sets forth a preferred example in connection with a document database of 500,000 documents selecting 10 documents most likely to meet the information need. Clearly, the algorithm may be used to provide the parameters for databases of other sizes, selection of other numbers of documents, and tolerance within other maximum error rates. Moreover, the algorithm may be modified to fit other examples in other situations, and, in fact, other algorithms are possible to define the sampling technique.

It may be desirable to employ the probability threshold technique described above with the statistical optimization selection described above. Hence, referring to Table I, the probability threshold may be set from the first sample requiring that documents selected during successive iterations also equal or exceed the probability threshold. As the processing continues, if the document of the first sample is ultimately replaced (that is, for a given iteration the probability of the first sample document is exceeded by the probabilities of at least the number of documents required by the iteration), a new threshold is established as the threshold of the new lowest document. Consequently, the probability threshold level continues to advance as documents are continued to be identified.

FIG. **10** is a flowchart of the steps of the statistical optimization selection technique of developing the probability threshold and document distribution optimization for the present invention.

More particularly, at step **150** the document distribution table of Table I is initialized to meet the criteria for error, numbers of documents sought, and collection size in accordance with the above-described software algorithm. At step **152**, the probability threshold value is initialized to 0 and the number of documents sought to be identified, D, is initialized to one. At step **154**, a document from the collection is scored utilizing the maximum score optimization technique, explained below in connection with FIG. **11**. At the same time, the number of documents processed since the previous document was scored is identified. At step **156**, a count is incremented identifying the total number of documents from the collection which had been processed.

Referring to Table I, if the first thirty documents of the collection contain no representations matching a concept of the query, the documents will not be scored because their probabilities would be 0.4. If the thirty-first document is the first document of the collection having representations which meet concepts of the query, that document is located and scored at step **154** using the maximum score optimizations described below. At the same time, a count of **31** is entered, representative of the number of documents processed (x_i). Since the thirty-first document is the only document in the result list, it is placed at the top of the result list.

At step **158**, the value from the table corresponding to D_i is compared against the number of documents x_i counted at step **156**. If the number of document, x_i, is smaller than the number D_i, the process continues to step **160**. At step **160**, each scored document is entered into the result list stored in the memory of the computer in descending order of probabilities. Thus, the document with the highest probability appears at the top of the result list whereas the document meeting the maximum score optimizations having the lowest probability is at the bottom of the list. In the initial iteration, x_i is 31 since thirty-one documents had been processed, and the value from Table I is 309 (corresponding to D_i=1).

Since the value from the table, 309, is greater than x_i, 31, the probability threshold is set at step **162** to the score for the Dth document in the result list, which in the example is the thirty-first document. At step **164**, the number of documents processed, x_i, is compared to the total number of documents in the collection, n_c, and if the number of documents processed is smaller than the number of documents in the collection, the process loops back through point **166** to return to step **154**. Any further documents which have probabilities less than the threshold probability (or which cannot mathematically achieve a probability greater than the probability threshold after calculation of less than all representation probabilities) are excluded (or not scored) at step **154**.

Assume document one hundred eighty has a probability greater than the probability threshold established by document thirty one. Hence, document one hundred eighty is identified at step **154** and inserted into the result list in probability order, which is greater than document thirty one. At step **156**, x_i is incremented to indicate the count, 180, of the number of documents thus far processed, which count is still smaller than 309, the number in Table I associated with D_i. Consequently, the sequence proceeds to step **160** to insert document one hundred eighty into the result list. At step **162** the probability threshold is set to the score of the Dth document in the result list. Since D_i is 1, the probability threshold is set to the score of document one hundred eighty.

Assume the next document having a probability greater than the probability threshold set by document one hundred eighty is document six hundred ten. Document six hundred ten is found and scored at step **154**. At step 156 the count x_i is incremented to 610, and since the value 309 from Table I is not greater than 610 at step 156, D_i is incremented by 1 at step **168** so that the new value from Table I to be

considered is 11,095. The process loops back to step 158 where the value 11,095 from Table I is found to be greater than 610. Hence the process continues to step **160** where document six hundred ten is inserted in the result list in probability order. At step **162** a new probability threshold equal to the Dth document in the result list is to be set. In this case, however, nothing occurs because D_i is now set to 2, meaning that both documents one hundred eighty and six hundred ten appear in the result list, and the probability threshold will continue to be set to the score of the document of the result list having the lowest probability, namely document one hundred eighty.

The process continues through the remainder of the database, incrementally increasing the value from Table I against which the document number is processed at step **158**, the process continuing until 10 documents are identified and all documents in the database have been processed. When this occurs, x_i equals n_c at step **164** and the final result list is retrieved at step **168**.

It might be advantageous, particularly where small document collections are to be searched and processing power is large, to perform the process of FIG. **10** for only a single iteration to find the document of the first sample having the highest probability and setting the probability threshold to the probability of that document for scoring the remainder of the document collection in the manner described above. Thus, the probabilities of documents added to the result list must exceed the initial probability threshold, at least until the preselected number of documents is added to the result list. Thereafter, the probability threshold is increased as additional documents having higher probabilities are added to the list and documents with the lowest probabilities are removed from the list.

In any event, if less than the preselected number of documents are ultimately identified to the result list, a new probability threshold may be established slightly below the probability of the document on the result list with the lowest probability and the entire collection re-scored as described above.

Maximum Score optimization

This technique is illustrated in the flow chart of FIG. **11**. More particularly, FIG. **11** illustrates the iterative loops for scoring documents employed at step **154** in FIG. **10**. Each document in the document database has a document number associated with it. The maximum score optimization commences with the concept i_1 in the query having the highest idf_i. A lower bound document number is chosen (such as the lowest document number in the database). The first document d_j whose document number is greater than the lower bound document number and which contains the concept i_1 is selected as a candidate document.

A remainder score is initialized to the maximum possible score less the value that document d_j scores for the concept i_1 being examined. Thus, the remainder score value represents the maximum score which each document which does not contain concept i_1 could achieve without concept i_1. The process continues by iterating through each of the concepts i_2, i_3, etc. The concepts are processed in descending order of concept idf_i value. As noted above, the concept with the highest idf_i is the concept which appears least frequently in the collection and is more likely to be a good discriminator than a concept which appears more often. The processing for each concept commences with the document having a document number greater than or equal to the lower bound document number.

In the processing, three conditions can occur.

1. If the document number for the current concept is equal to that of the candidate document, the candidate document contains the concept and no change is made to the maximum score. Instead, the process continues to the next concept.

2. If the document number for the current concept is greater than that of the candidate document, the current document does not contain the concept and the value of the current concept is subtracted from the maximum score for the candidate document and the remainder score is adjusted. If the maximum score is still high enough that the candidate document might still be selected, the processing will continue to the next concept. If not, the candidate document is discarded and the processing starts over with the next higher document number as the candidate document.

3. If the document number for the current concept is less than that of the candidate document, a document exists with a lower number which must be evaluated before continuing with the candidate document.

The remainder score tabulated for each document represents the maximum score that document can achieve based on the concepts processed up to that point and the possibility that it contains all the subsequent concepts. As each concept is processed, the remainder score for the document is reduced by the value of the concept for each document in which the concept does not appear. In considering the remainder score, two possibilities exist.

1. If the remainder score is less than the minimum document score necessary to remain in the result list, then that document, and all other documents up to the candidate document number, can be discarded, since it is not possible for any of them to achieve a document score high enough to remain in the result list. In this situation, the next document number which is greater than or equal to the candidate document number is selected for the concept and the processing continues as described above.

2. If the remainder score is not less than the minimum document score necessary to remain in the result list, then the document is considered as a candidate for the result list. In this case, the document score for the document is set to the current remaining score and the candidate document number is reset.

The process continues until a candidate is found having a maximum possible score greater than the probability threshold required to remain in the result list.

The process of the maximum score optimization may be explained with reference to the flowchart of FIG. **11**. At step **180** the lower bound document number, probability threshold (from step **152** or **162** in FIG. **10**) and the maximum possible score are inputted. For the initial iteration for a given document, the probability threshold is initialized to 0 at step **152** in FIG. **10** and the maximum possible score is initialized. The lower bound document number is set to the first document in the database desired to be reviewed. At step **182**, the first document having a document number greater than or equal to the lower bound document number and which contains the concept having the highest idf_i is identified as a candidate document. Thus, the document number is identified for the first document containing the concept. At step **184**, the remainder score for all other documents having a lower number is initialized to be equal to the maximum possible score less the incremental concept value from the missing concept i_1 having the highest idf_i. At step **186**, a decision is made as to whether all the concepts have been processed, and if they have not, the current concept is set to the concept i_2 whose idf_i is next highest in value below the first concept i_1, at step **188**. At step **190**, the document number is set to the document number of the next document greater than or equal to the lower bound document number for the current (second) concept i_2. At step **192**, if

the document number of the document containing the concept is less than the current candidate document number, then the decision is made at step **194** whether the remainder score is smaller than the probability threshold initialized at step **152** or set at step **162** in FIG. **10**. If the remainder score is smaller than the minimum probability threshold, then the lower bound document number is set to the current candidate document number and the document number of the next document containing the concept i_2 currently being processed is set to the next document number greater than or equal to the current lower bound document number for the current concept. The concept incremental value is subtracted at step **200** from the remainder score. If, at step **194**, the remainder score is greater than or equal to the probability threshold, then the candidate document number is set, at step **202**, to the document number of the next document containing the concept, and the candidate document score is set, at step **204**, to the remainder score. The process then continues to step **200** to subtract the concept incremental value from the remainder score for the documents not containing the concept.

If at step **192** the document number containing the concept is greater than or equal to the candidate document number, then the process continues directly to step **200** where the concept incremental value is subtracted from the remainder score for the documents not containing the concept.

At step **206**, if the document number containing the concept is equal to the candidate document number, then the candidate document is found to contain the concept, and the process returns to step **186** and processes through the loop again for the next concept. If the document number containing the concept is not equal to the candidate document number, then the concept incremental value is subtracted from the candidate document score at step **208**. If the resulting candidate document score is greater than the probability threshold, the process loops back through step **186** again. On the other hand, if the candidate document score is not greater than the probability threshold, the lower bound document number is set to the candidate document number plus **1** and the process reloops to step **182**.

If a candidate document loops through the process of FIG. **11** through all of the concepts of the query, and the document score is greater than the probability threshold at step **210**, step **186** identifies that all concepts have been processed and returns the document at step **214** for insertion into the full result list in sorted order at step **156** in FIG. **10**. The process terminates for a given threshold value only when a candidate is found, after all concepts have been examined, which has a maximum possible score greater than the probability threshold required to remain in the result list. The process iterates through the loops illustrated in FIG. **10** until the required number of documents for the result list is identified. The documents may then be retrieved from database using the result list at step **170**, the scoring of each document occurring through the iterations of the loops of FIG. **11**.

It may be desirable to incorporate certain relational constraints on the placement of documents into the result list. As one example, it might be desirable to limit the search output to documents dated after a given date. Suffice it to say that such a constraint can be imposed on the document retrieval system in a manner well known in the art.

Document Retrieval

FIGS. **12** and **13** are flowcharts detailing the construction and evaluation of an inference network, FIG. **12** being a detailed flowchart for constructing the query network **12** and FIG. **13** being a detailed flowchart for evaluation the query network in the context of the document network **10**. As heretofore described, an input query written in natural language is loaded into the computer, such as into a register therein, and is parsed (step **220**) compared to the stopwords in database **222** (step **224**) and stemmed at step **226**. The result is the list **42** illustrated in FIG. **4**. Using synonym database **228**, the list is compared at step **230** to the synonym database and synonyms are added to the list. As will be explained hereinafter, the handling of synonyms may actually occur after handling of the phrases. Citations are located at step **232** as heretofore described. More particularly, a proximity relationship is established showing the page number within five words of the volume number, without regard to the reporter system employed. The handling of citations, like the handling of synonyms, may be accomplished after phrase resolution, if desired.

Employing phrase database **234**, a decision is made step **236** as to whether or not phrases are present in the query. If phrases are present, a comparison is made as step **240** to identify phrases. At step **242** a determination is made as to whether successive phrases share any common term(s) (an overlap condition). More particularly, and as heretofore described, terms which are apparently shared between successive phrases are detected at step **242**. At step **244** a determination is made as to which phrase is the longer of the two phrases, and the shared term is included in the longer phrase and excluded from the shorter phrase. As a result of deleting the shared term from the shorter phrase, the resulting shorter phrase may not be a phrase at all, in which case the remaining term(s) are simply handled as stemmed words. On the other hand, if the two phrases are of equal length, then the shared term is accorded to the first phrase and denied to the second phrase.

After overlap conflict is resolved at step **244**, the resulting phrase substitution occurs at step **246**. The process loops back to step **236** to determine if phrases are still present, and if they are the process repeats until no further phrases are present. At step **238**, all duplicate terms are located, mapped, counted and removed, with a count V representing the number of duplicate terms removed. Thus, the search query illustrated at block **46** in FIG. **4** is developed.

As heretofore described, the handling of synonyms and citations may occur after resolution of the phrases, rather than before.

As illustrated in FIG. **13**, the resulting search query is provided to the document network where, at step **250** the number of terms T is counted, at step **252** i is set to **0** and at step **254** 1 is added to i. Using document database **256** which also contains the text of the documents, the inverse document frequency (idf_i) is determined and the probability estimate (tf_i) is determined at step **258**. As noted above, both tf_i and idf_i are calculated from addresses, document numbers and offset data in the word index of the document database. The estimated inverse document frequency (idf_i) is also added to the database by a temporary memory or register. The component probability is determined at step **260** as heretofore described and is accumulated with other component probabilities at step **262**. At step **264** a determination is made as to whether or not i equals T

(where T is the number of terms in the search query). If all of the terms have not been compared to the database, the process is looped, adding 1 to i and repeated for each term until i equals T at step **264**. As heretofore described, when terms having duplicates deleted from the input query are processed at step **258**, the probability for such terms is multiplied by the number of duplicates deleted, thereby weighing the probability in accordance with the frequency

of the term in the original input query. Consequently, at step **266**, it is necessary to divide the accumulated component probability for the document by V+T (where V is the number of duplicate terms deleted from the input query) to thereby normalize the probability. The probability for each document is stored at step **268** and the process repeated at step **270** for the other documents. At step **272** the documents are ranked in accordance with the determined probabilities, and the top ranked documents are printed out or displayed at step **274**.

As previously described, the scan technique may be a concept-based scan, rather than the document-based scan described. Further, as previously described, the scan may be aborted after less than complete scan of any given document if the probabilities result in a determination that the document will not reach the cutoff for the D top-ranked documents to be displayed or printed.

While the present invention has been described in connection with a time-shared computer system shown in FIG. **3** wherein search queries are generated by PC computers or dumb terminals for transmission to and time-shared processing by a central computer containing the document network, it may be desirable in some cases to provide the document network (with or without the document text database) to the user for direct use at the PC. In such a case, the document database would be supplied on the same ROM **24** as the databases used with the search query, or on a separately supplied ROM for use with computer **20**. For example, in the case of a legal database, updated ROMs containing the document database could be supplied periodically on a subscription basis to the user. In any case, the stopwords, phrases and key numbers would not be changed often, so it would not be necessary to change the ROM containing the databases of stopwords, phrases and key numbers.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

What is claimed is:

1. In a computer system for identifying a predetermined number of documents of a document collection containing representations that have high probabilities of matching a query containing a plurality of concepts, in which the system has a database containing identifications of documents in the document collection and defining a plurality_ of representations representing the contents of the documents, the collection comprising a plurality_ of documents, and query means for defining the query, apparatus comprising:

sample selection means for iteratively selecting successive samples of a plurality of documents from the collection, each sample containing fewer documents than the entire collection and each successive sample containing documents different from each previous sample;

processing means responsive to the sample selection means for calculating, during each iteration, probabilities that documents contained in the sample contain representations that match the query and for identifying a preselected number of documents having the highest probabilities, the documents being identified during an iteration from a group consisting of the respective sample of documents and the documents identified during the next previous iteration, the preselected number being different for each iteration and no greater than the predetermined number; and

output means outputting the identifications of the predetermined number of documents identified by the processing means.

2. The apparatus according to claim **1** further including threshold setting means responsive to the processing means for setting a probability threshold equal to the probability of a first identified document.

3. The apparatus according to claim **2** including determining means operable during each respective iteration and responsive to the identification of the preselected number of documents by the processing means to determine if an additional document of the respective sample has a probability greater than the probability threshold, the processing means being responsive to the determining means identifying an additional document having a probability greater that the probability threshold to replace the previously-identified document having the lowest probability by the additional document, and the threshold setting means being responsive to the processing means to reset the probability threshold to the probability of the identified document having the new lowest probability.

4. The apparatus according to claim **1** wherein the preselected number is equal to the number of the respective iteration, and the predetermined number is equal to the number of the last iteration.

5. The apparatus according to claim **1** including estimating means responsive to the processing means for estimating a maximum probability for a second document different from the first document based on a partially calculated probability for the second document and an assumption that the representations in the second document match the concepts of the query for which probabilities have not been calculated, the processing means being responsive to the estimating means to calculate partial probabilities that representations in the second document match concepts of the query until either the estimated maximum probability does not at least equal the threshold or the probability is calculated for all the concepts in the query.

6. The apparatus according to claim **5** wherein the output means includes a result list ranking the identified documents in probability order, the threshold setting means being responsive to the result list to reset the probability threshold equal to the probability of the document lowest on the result list.

7. The apparatus according to claim **1** wherein the processing means includes a result list ranking the identified documents in probability order.

8. A system for identifying documents matching a comprising:

a memory containing a database containing identification of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, the database further containing indications of the frequencies of occurrence of documents containing first representations in the collection;

computer means responsive to a query defining a plurality of concepts, the computer means including

matching means for matching the concepts to representations,

estimating means for estimating the frequency of occurrence of documents containing a second selected representation in the collection, the second selected representation being different from any of the first representations, the estimating means including

sample selection means for selecting a sample comprising a plurality of documents from the collection,

the sample containing fewer documents than the entire collection;

frequency identifying means responsive to the sample selection means for identifying the frequency of occurrence of documents containing the second selected representation in the selected sample of documents;

processor means responsive to the memory means and to the frequency identifying means for calculating a maximum and a minimum probable frequency of occurrence of documents containing the second selected representation in the collection; and

selection means responsive to the processor means for selecting the midpoint, of the maximum and minimum probable frequencies as the estimated frequency of occurrence of the second selected representation;

retrieval means for selecting documents meeting the query based on the frequencies of occurrence of documents containing first representations which match the concepts and the estimated frequencies of occurrence of documents containing second representations which match the concepts, and

output means responsive to the retrieval means and the memory for outputting identifications of the selected documents.

9. The system according to claim **8** wherein the processor means includes means for identifying if the difference between the maximum and minimum probable frequencies is within a preselected limit, and further including adjusting means responsive to the processor means for adding additional documents from the collection to the sample of documents if the calculating difference between the maximum and minimum probable frequencies exceeds the preselected limit.

10. The system according to claim **8** where the processor means calculates the maximum probable frequency, f_{max}, and the minimum probable frequency, f_{min}, in accordance with relationships based on the number of gaps between documents in the sample containing the second selected representation (n_i), the number of documents in the collection (n_c), and the number of documents in the sample (x_i).

11. The system according to claim **10** where f_{max} and f_{min} are calculated accordance with the relationships

$$f_{max} = n_i + \frac{n_i(n_c - x_i)}{x_i - zs_i\sqrt{n_i}} + z\sqrt{\frac{n_i(n_c - x_i)}{x_i}}$$

and

$$f_{min} = n_i + \frac{n_i(n_c - x_i)}{x_i + zs_i\sqrt{n_i}} - z\sqrt{\frac{n_i(n_c - x_i)}{x_i}}$$

where

s_i is the greater of x_i/n_i or the standard deviation of the n_i gaps, and

z is the standard critical value for normal distribution for a preselected reliability.

12. In a system for identifying documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection compromising a plurality of documents and the database containing a frequency of occurrence of documents containing each of at least some of the representations in the collection of documents, query means for defining a query containing a plurality of concepts, matching means for matching concepts to representations means for selecting documents meeting the query based on frequencies of occurrence of documents in the collection containing representations matching the concepts, and output means responsive to the means for selecting documents for outputting identifications of selected documents, the improvement comprising a process of estimating the frequency of occurrence of documents containing a representation in the collection of documents for which the database does not contain a frequency of occurrence, comprising:

identifying, on the basis of concepts in the query, a representation for which the database does not contain a frequency of occurrence;

selecting a sample comprising a plurality of documents from the collection, the sample containing fewer documents than the entire collection;

identifying the frequency of occurrence of documents containing the identified representation in the selected sample of documents;

calculating a maximum and a minimum probable frequency of occurrence of documents containing the identified representation in the collection; and

selecting a midpoint of the maximum and minimum probable frequencies as the estimated frequency of occurrence of documents containing the identified representation,

whereby the means for selecting documents meeting the query is responsive to the frequencies of occurrence in the database of documents in the collection containing representations matching the concepts and to estimated frequencies of occurrence to select documents in the collection containing representations matching the concepts.

13. The process according to claim **12** further including identifying whether the difference between the maximum and minimum probable frequencies is within a preselected limit, and adding additional documents to the sample from the collection if the calculated difference between the maximum and minimum probable frequencies exceeds the preselected limit.

14. The process according to claim **13** where the preselected limit is 0.05.

15. The process according to claim **12** where the maximum probable frequency, f_{max}, and the minimum probable frequency, f_{min}, are calculated in accordance with the relationships

$$f_{max} = n_i + \frac{n_i(n_c - x_i)}{x_i - zs_i\sqrt{n_i}} + z\sqrt{\frac{n_i(n_c - x_i)}{x_i}}$$

and

$$f_{min} = n_i + \frac{n_i(n_c - x_i)}{x_i + zs_i\sqrt{n_i}} - z\sqrt{\frac{n_i(n_c - x_i)}{x_i}}$$

where

n_i is the number of gaps between documents in the sample containing the selected representation,

n_c is the number of documents in the collection,

x_i is the number of documents in the sample,

s_i is the greater of x_i/n_i or the standard deviation of the n_i gaps, and

z is the standard critical value for normal distribution for a preselected reliability.

16. The process according to claim **15** where the selected representation contains a plurality of terms, the method

including setting f_{min} equal to n_t if the calculated f_{min} is smaller than n_t, setting f_{max} equal to $n_t+(n_c31\ x_i)$ if the calculated f_{max} is smaller than zero or smaller than n_t, and setting f_{max} equal to an a priori maximum if the calculated f_{max} is greater than the a priori maximum.

17. The process according to claim 16 wherein the selected representation is a synonym represented by a plurality of terms, and wherein the a priori maximum is equal to the sum of all frequencies of occurrence of documents in the collection containing a term of the synonym, said method including setting f_{min} equal to an a priori minimum if the calculated f_{min} is smaller than the a priori minimum, where the a priori minimum is equal to the frequency of occurrence of documents containing the term of the synonym appearing in the greatest number of documents in the collection.

18. The process according to claim 16 wherein the selected representation is a phrase containing a plurality of terms, and the a priori maximum is equal to the frequency of occurrence of documents containing the term of the phrase appearing in the least number of documents in the collection.

19. The process according to claim 15 where the preselected reliability is 0.995 and z is 2.8070.

20. The process according to claim 12 wherein the midpoint selected between the maximum and minimum probable frequencies is the mean of the maximum and minimum probable frequencies.

21. In a computer system for identifying documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, and query means for defining a query containing a plurality of concepts, apparatus for identifying documents of the document collection containing representations that match the query containing a plurality of concepts, the apparatus comprising:

processing means for calculating probabilities that documents match the query and for identifying a first document having a calculated probability;

threshold setting means responsive to the processing means for setting a probability threshold equal to the probability of the first document;

estimating means responsive to the processing means for estimating a maximum probability for a second document different from the first document based on a partially calculated probability and an assumption that the representations in the second document match the concepts of the query for which probabilities have not been calculated;

the processing means being responsive to the estimating means to calculate partial probabilities that representations in the second document match concepts of the query until either the estimated maximum probability for the second document does not at least equal the probability threshold or the probability is calculated for all the concepts in the query;

the estimating means being further responsive to the processing means ceasing or completing the calculation of the probability for the second document to estimate a maximum probability for a third document different from the first and second documents; and

output means responsive to the processing means for outputting identifications of only documents whose probability is calculated for all concepts in the query.

22. The apparatus according to claim 21 wherein the output means includes a result list identifying in probability order, up to a predetermined number of documents whose probability is calculated for all concepts in the query, the threshold setting means being responsive to the result list to reset the probability threshold equal to the probability of the document lowest on the result list.

23. Apparatus according to claim 21 wherein the threshold setting means is responsive to the processing means calculating the probability for the second document for all the concepts in the query to set the probability threshold equal to the probability of the second document.

24. The apparatus according to claim 21 wherein the output means includes a result list identifying in probability order, up to a predetermined number of documents whose probability is calculated for all concepts in the query.

25. A document identification system for identifying a predetermined number of documents matching a query, comprising:

a read-only memory containing a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of documents in the document collection, the collection comprising a plurality of documents;

query means for defining the query containing a plurality of concepts;

computer means responsive to the query containing a plurality of concepts, the computer means including matching means for matching the concepts to representations;

sample selection means for iteratively selecting successive samples of a plurality of documents from the collection for examination, each sample containing fewer documents than the entire collection, and each successive sample containing documents different from each previous sample;

processing means responsive to the sample selection means for calculating, during each iteration probabilities that documents contained in the sample contain representations that match the query and for identifying up to a preselected number of documents having the highest probabilities, the documents being identified during each iteration from a group consisting of the respective sample of documents and the documents identified during the next previous iteration, the preselected number being different for each iteration and no greater than the predetermined number; and

output means outputting identifications of the predetermined number of documents identified by the processing means.

26. The system according to claim 25 further including threshold setting means responsive to the processing means for setting a probability threshold equal to the probability of a first identified document.

27. The system according to claim 26 including determining means operable during each respective iteration and responsive to the identification of the preselected number of documents by the processing means to determine if an additional document of the respective sample has a probability grater than the probability threshold, the processing means being responsive to the determining means identifying an additional document having a probability greater than the probability threshold to replace the previously-identified document having the lowest probability with the additional document, and the threshold setting means is responsive to the processing means to reset the probability threshold to the

probability of the identified document having the new lowest probability.

28. The system according to claim 25 including estimating means responsive to the processing means for estimating a maximum probability for a second document different from the first document based on a partially calculated probability for the second document and an assumption that the representations in the second document match the concepts of the query for which probabilities have not been calculated, the processing means being responsive to the estimating means to calculate partial probabilities that representations in the second document match concepts of the query until either the estimated maximum probability for the second document does not at least equal the threshold or the probability is calculated for all the concepts in the query.

29. The system according to claim 28 wherein the output means includes a result list ranking the identified documents in probability order, the threshold setting means being responsive to the result list to reset the probability threshold equal to the probability of the document lowest on the result list.

30. The system according to claim 25 wherein the output means includes a result list ranking the identified documents in probability order.

31. A document identification system for identifying documents matching a query, comprising:

a read-only memory containing a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of documents in a document collection, the collection comprising a plurality of documents, the database further containing indications of the frequencies of occurrences of a plurality of representations in the documents;

query means for defining the query containing a plurality of concepts;

computer means responsive to the query, the computer means including matching means for matching the concepts to representations;

calculating means for calculating the probabilities that documents meet the query based on the frequencies of occurrence of representations in the respective documents which match the concepts;

processing means responsive to the calculating means for identifying a first document contained in the sample having the highest calculated probability;

threshold setting means responsive to the processing means for setting a probability threshold equal to the probability of the first document;

estimating means responsive to the calculating means for estimating a maximum probability for a second document different from the first document based on a partially calculated probability for the second document and an assumption that the representations in the second document match the concepts of the query for which probabilities have not been calculated,

said calculating means being responsive to the estimating means to calculate partial probabilities that representations in the second document match concepts in the query until either the estimated maximum probability for the second document does not at least equal the probability threshold or the probability is calculated for all concepts in the query,

the estimating means being further responsive to the calculating means ceasing or the completing the calculation of the probability for the second document to estimate a maximum probability for a third document different from the first and second documents; and

output means responsive to the processing means for outputting identifications of only documents whose probability is calculated for all concepts in the query.

32. The document identification system according to claim 31 wherein said output means includes a result list responsive to the calculating means to identify in probability order up to a predetermined number of those documents whose probability is calculated for all concepts in the query, said threshold setting means being responsive to the result list to reset the probability threshold equal to the probability of the document lowest on the result list.

33. In a computer system for identifying documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, and query means for defining a query containing a plurality of concepts, a process of identifying a predetermined number of documents of the document collection containing representations that have high probabilities of matching the query containing a plurality of concepts, the process comprising:

iteratively selecting successive samples of a plurality of documents from the collection for examination, each sample containing fewer documents than the entire collection, and each successive sample containing documents different from each previous sample;

calculating the probabilities that documents contained in the sample contain representations that match the query;

identifying, during each iteration, a preselected number of documents having the highest probabilities, the documents being selected from a group consisting of a respective sample of documents and the documents identified during the next previous iteration, the preselected number being different for each iteration and no greater than the predetermined number; and

outputting identifications of the predetermined number of identified documents upon completion of the last iteration.

34. The process according to claim 33 including setting a probability threshold to the probability of the identified document having the lowest probability of all identified documents, and during each respective iteration and after the preselected number of documents has been identified, determining if an additional document of the respective sample has been identified having a probability greater than the probability threshold, and if so, replacing the previously-identified document having the lowest probability with the additional document and resetting the probability threshold to the probability of the identified document having the new lowest probability.

35. The process according to claim 33 wherein the preselected number is equal to the number of the respective iteration, and the predetermined number is equal to the number of the last iteration.

36. The process according to claim 33 including setting a probability threshold equal to the probability of a first document, estimating a maximum probability for a second document different from the first document based on a partially calculated probability for the second document and an assumption that the representations in the second document match the concepts of the query for which probabilities have not been calculated, and calculating partial probabili-

ties that representations in the second document match concepts in the query until either the estimated maximum probability for the second document does not at least equal the threshold or the probability is calculated for all the concepts in the query.

37. The process according to claim **36** including ranking the identified documents in probability order, and resetting the probability threshold equal to the probability of the document lowest on the list.

38. The process according to claim **33** including ranking the identified documents in probability order.

39. In a computer system for identifying documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, and query means for defining a query containing a plurality of concepts, a process of identifying documents of the document collection containing representations that match the query containing a plurality of concepts, the process comprising:

computing the full probability that a first document matches the concepts in the query;

setting a probability threshold equal to the full probability of the first document;

calculating a partial probability that a second document matches some but not all concepts in the query;

estimating a maximum probability for the second document based on the calculated probability and an assumption that the representations in the document match the concepts of the query for which probabilities have not been calculated;

repeating the steps of calculating and estimating for additional query concepts until either the estimated maximum probability tier the second document is not as large as the probability threshold or the full probability of the second document is calculated for all concepts in the query;

repeating the repetitive steps of calculating and estimating for a third document different from the first and second documents; and

outputting identifications of only documents having a full probability at least as great as the probability threshold.

40. The process according to claim **39** wherein a predetermined number of documents of the document collection is identified and wherein documents whose probabilities are calculated for all concepts in the query are identified to a result list in probability order, up to said predetermined number, said process further including resetting the probability threshold equal to the probability of the document lowest on the result list.

41. In a system identifying a predetermined number of documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, query means for defining containing a plurality of concepts, means for determining a probability that a document meets the query based on matches of representations in the document and concepts in the query, and output means for outputting the identifications of documents having a probability at least as great as a probability threshold, apparatus for establishing the probability threshold comprising:

sample selection means for iteratively selecting successive samples of a plurality of documents from the collection for examination, each sample containing fewer documents than the entire collection and each successive sample containing documents different from each previous sample;

calculating means for calculating probabilities that documents contained in the sample contain representations that match the query;

processing means responsive to the sample selection means to identify, during each iteration, up to a preselected number of documents having the highest probabilities, the documents being identified during each iteration from a group consisting of a respective sample of documents end the documents identified during the previous iteration; and

threshold setting means responsive to the processing means for setting the probability threshold to the probability of the identified document having the lowest probability.

42. The apparatus according to claim **41** including determining means operable during each respective iteration and responsive to the identification of the preselected number of documents by the processing means to determine if the processing means identifies an additional document of the respective sample having a probability greater than the probability threshold, the processing means being responsive to the determining means to replace the previously-identified document having the lowest probability by the additional document, and the threshold setting means is responsive to the processing means to reset the probability threshold to the probability of the identified document having the new lowest probability.

43. The apparatus according to claim **41** wherein the preselected number is equal to the number of the respective iteration.

44. In a system for identifying a predetermined number of documents matching a query, in which the system has a database containing identifications of documents in a document collection and defining a plurality of representations representing the contents of the documents, the collection comprising a plurality of documents, query means for defining a query containing a plurality of concepts, means for determining a probability that a document meets the query based on a match of representations in the document and concepts in the query, and output means for outputting the identifications of documents having a probability at least as great as a probability threshold, a process for establishing the probability threshold comprising:

iteratively selecting successive samples of a plurality of documents from the collection for examination, each sample containing fewer documents than the entire collection, and each successive sample containing documents different from each previous sample;

calculating probabilities that documents in the sample contain representations that match the query;

identifying, during each iteration, up to a preselected number of documents having the highest probabilities, the documents being identified during each iteration from a group consisting of a respective sample of documents and the documents identified during the next previous iteration; and

setting the probability threshold to the probability of the identified document having the lowest probability.

45. The process according to claim **44** including during each respective iteration and after the preselected number of documents has been identified, determining if an additional document of the sample has been identified having a prob-

ability greater than the probability threshold, replacing the previously-identified document having the lowest probability by the additional document, and resetting the probability threshold to the probability of the identified document having the new lowest probability.

46. The process according to claim 44 wherein the preselected number is equal to the number of the respective iteration.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,488,725

DATED : January 30, 1996

INVENTOR(S) : HOWARD R. TURTLE, GERALD J. MORTON, F. KINLEY LARNTZ

It is certified that error appears in the above-indentified patent and that said Letters Patent is hereby corrected as shown below:

After "References Cited U.S. PATENT DOCUMENTS", insert -- 4,241,402 12/1980 Mayper, Jr. et al. 364/200; 4,270,182 05/1981 Asija 364/900; 4,358,824 11/1982 Glickman et al. 364/200; 4,471,459 09/1984 Dickinson et al. 364/900; 4,499,553 02/1985 Dickinson et al. 364/900; 4,580,218 04/1986 Raye 364/300; 4,670,848 06/1987 Schramm 364/513; 4,688,195 08/1987 Thompson et al. 364/513; 4,706,212 11/1987 Toma 364/900; 4,787,035 11/1988 Bourne 364/300; 4,823,306 04/1989 Barbic et al. 364/900; 4,839,853 06/1989 Deerwester et al. 364/900; 4,862,408 08/1989 Zamora 364/900; 4,868,750 09/1989 Kucera et al. 364/419; 4,914,590 04/1990 Loatman et al. 364/419; 4,918,588 04/1990 Barrett et al. 364/200; 4,931,935 06/1990 Ohira et al. 364/419; 4,972,349 11/1990 Kleinberger 364/900; 4,974,191 11/1990 Amirghodsi et al. 364/900; 4,991,087 02/1991 Burkowski et al. 364/900; 5,099,425 03/1992 Kanno:Yuji et al. 364/419; 5,117,349 05/1992 Tirfing et al. 395/600 --

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,488,725
DATED : January 30, 1996
INVENTOR(S) : HOWARD R. TURTLE, GERALD J. MORTON, F. KINLEY LARNTZ

It is certified that error appears in the above-indentified patent and that said Letters Patent is hereby corrected as shown below:

After "References Cited Other Publications" delete "Porter, An Algorithm for Suffix Skipping", Program, vol. 15, pp. 130-137 (1980).", insert -- Porter, "An Algorithm for Suffix Skipping", Program, vol. 14, pp. 130-137 (1980).--

Col. 25, line 39, delete "optomization", insert --Optomization--

Col. 29, line 47, after "plurality", delete "_"

Col. 29, line 49, after "plurality", delete "_"

Col. 30, line 48, before "comprising", insert --query--

Col. 31, line 33, delete "calculating", insert --calculated--

Col. 31, line 63, delete "compromising", insert --comprising--

Col. 34, line 61, delete "grater", insert --greater--

Col. 38, line 13, delete "end", insert --and--

Signed and Sealed this

Twenty-eighth Day of May, 1996

Attest:

BRUCE LEHMAN

Attesting Officer — *Commissioner of Patents and Trademarks*

US 1006211004P1

Creation date: 07-23-2004
Indexing Officer: DSHEA - DUNCAN SHEA
Team: OIPEBackFileIndexing
Dossier: 10062110

Legal Date: 07-13-2004

No.	Doccode	Number of pages
1	CTNF	8
2	892	1

Total number of pages: 9

Remarks:

Order of re-scan issued on .....